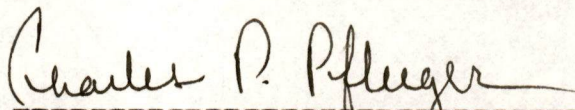


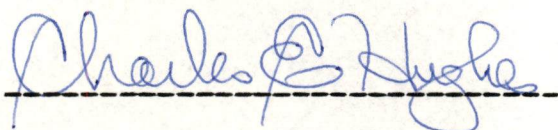
To the Graduate Council:

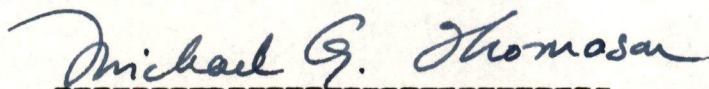
I am submitting herewith a thesis written by Anthony Michael Peters entitled "A Virtual Memory for the Extended Interpreter of ASSIST-V." I recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.



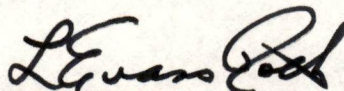
Charles F. Pfleeger, Major Professor

We have read this thesis
and recommend its acceptance:





Accepted for the Council:



Vice Chancellor
Graduate Studies and Research

Thesis
79
.P484
cop. 2

A VIRTUAL MEMORY FOR THE
EXTENDED INTERPRETER
OF ASSIST-V

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Anthony Michael Peters

December 1979

1405017

TO MY WIFE AND CHILDREN

ACKNOWLEDGMENTS

I wish to express my appreciation to Dr. Charles Pfleeger and Dr. Charles Hughes for their time, guidance and friendship given during the writing of this thesis as well as throughout my course of study at The University of Tennessee. I am also grateful to Dr. David Straight and Dr. Michael Thomason for their time and consideration as members of my thesis committee.

ABSTRACT

This paper describes the addition of a Dynamic Address Translation feature to ASSIST-V. ASSIST-V is a program which provides an environment for the implementation, testing and evaluation of systems software for the IBM System/360 series of computers. The implementation of the Dynamic Address Translation feature will assist in the teaching of virtual memory and paging techniques. The Dynamic Address Translation feature portion of ASSIST-V will provide an environment for implementation and testing of virtual memory systems software for the IBM System/370 series of computers. This capability is achieved by simulating all relevant features of the System/370. In particular, the actual translation process, control registers, control modes of operation, and translation interrupts are added to ASSIST-V.

TABLE OF CONTENTS

SECTION	PAGE
I. PROJECT OVERVIEW	1
Objectives	1
Implementation Environment	2
II. DYNAMIC ADDRESS TRANSLATION USER'S GUIDE	4
Purpose and Scope of this Guide.	4
Introduction to ASSIST-V	4
Introduction to Virtual Memory	8
Virtual Storage in the System/370.	10
Using the Virtual Memory of the Extended Interpreter of ASSIST-V.	15
ASSIST-V Address Translation	19
Translation Program Exceptions	23
An Example	24
III. PROGRAM LOGIC MANUAL	59
Purpose.	59
SYSGEN Parameters.	59
Dynamic Address Translation Overview	60
In-line Additions to ASSIST-V to Support DAT	60
Support Routines for DAT	63
Implementation of Control Registers.	66
Changes to ASSIST-V for DAT.	68
Bugs in ASSIST-V Affecting DAT	69
Possible Additions to DAT.	70
LIST OF REFERENCES	72
VITA	75

LIST OF FIGURES

FIGURE	PAGE
1. IBM Address Structure.	11
2. Address Translation.	13
3. Object Deck Format	17
4. Sample JCL for Separate Assemblies	18
5. ASSIST-V Address Structure	20
6. Control Register 1	21
7. Segment Table Entry.	22
8. Page Table Entry	23
9. User Test Program.	25

I. PROJECT OVERVIEW

1. OBJECTIVES

Although much can be learned through the reading of appropriate textbooks, programming experience is still necessary to complete the education of the systems programmer. Such activity generally requires the dedication of a general purpose machine. Such dedication is generally not feasible on mainframe computers. ASSIST-V, a software tool designed for use in the teaching of operating systems, file management, and machine organization courses provides an environment for the implementation, testing, and evaluation of systems software for the IBM System/360 series of computers. However, the current state-of-the-art of computer technology requires that a virtual memory environment be added to any such educational tool.

Virtual memory is a method of automatically overlaying programs without the knowledge of the programmer. A programmer may code a program with regard to only the size constraints of the address space of a particular instruction set rather than the actual size of physical memory. The automatic overlay process, called paging, gives the programmer the illusion that a large, continuous, linear main memory, the same size as the address space exists when, in fact, the main memory may be considerably smaller than the address space. This simulation of a large main memory is not detected by the program running under such a system.

Programs are executed by mapping the logical addresses, or the ones to which a program can refer, to actual, physical memory locations. This address mapping process is known as Dynamic Address Translation in the IBM System/370 machines. This thesis has added a simulation of the Dynamic Address Translation feature to ASSIST-V. With ASSIST-V it is now possible to interrupt a program at an arbitrary moment, record it and its data on virtual external storage devices, and at a later time return the program to any simulated main-storage locations for resumption of execution.

2. IMPLEMENTATION ENVIRONMENT

The ASSIST-V simulator was written as a major extension to an existing program called ASSIST. ASSIST is a student oriented assembler/interpreter for IBM 360-370 machines. It was designed and implemented by Drs. John R. Mashey and Graham M. Campbell, both of The Pennsylvania State University (Mashey, Campbell and Forney, 1972, Mashey, 1973, and Overbeek and Singletary, 1976). The present version, maintained by the authors and distributed by The Pennsylvania State University, is in use at over 200 universities in the United States and abroad.

ASSIST-V is an extension to ASSIST. The assembler was preserved virtually intact, and extensive additions were made to the interpreter. Since the assembler was unchanged, no change to assembly speed has occurred; execution speed is approximately 4300 instructions per second on an IBM 360

model 65. The version of ASSIST-V as described in [3] requires 74K bytes of storage plus user area.

The Dynamic Address Translation feature requires 4K bytes of storage in addition to the 74K required for ASSIST-V. Page size of the Dynamic Address Translation is fixed at 256 bytes to provide a workable page size in a student environment and because the standard 2K/4K page size of the System/370 is not compatible with the track size of ASSIST-V virtual disks. ASSIST-V has remained intact with the exception of a reduction in size of main memory blocks, changing from 2K to 256 bytes, to provide protection for 256 byte page frames.

II. DYNAMIC ADDRESS TRANSLATION USER'S GUIDE

1. PURPOSE AND SCOPE OF THIS GUIDE

The purpose of this guide is to define to the user of ASSIST-V the concepts of the Dynamic Address Translation feature of ASSIST-V and to assist in its use. To use this guide, you should be familiar with virtual memory and ASSIST-V, inclusive of the I/O package. There is a brief introduction to ASSIST-V in the following section. For a more detailed explanation the user is referred to ASSIST-V: An Environmental Simulator for the IBM 360 System Software Development [4]. Other documents that you may need to consult are the IBM System/370 Principles of Operation [10,] which gives information on the hardware of the System/370 and the IBM Introduction to Virtual Storage Systems [6], which gives information on the System/370 virtual memory; that material will be assumed in this guide.

2. INTRODUCTION TO ASSIST-V

Initial Machine Configuration

The user of ASSIST-V prepares an "operating system" and a series of one or more test programs for that system. The term "operating system" in this context means one or more supervisory routines to oversee the execution of the test programs. The supervisory routine may be as simple as the interrupt handler for a single type of interrupt, or it may be as complex as a multiprogramming operating system. The

operating system might even be a loader program that reads the test programs from the input stream, or the loader might read an operating system that in turn reads test programs.

This operating system and its test program(s) are submitted to ASSIST-V for assembly and execution. ASSIST-V assembles the operating system and relocates the resultant object code as if it were placed beginning at location 0 of an IBM 360. The operating system should include code to define the reserved storage locations for new PSW's for each interrupt type being handled. An initial value should also be assigned for the interval timer at location X'50', if timer services are used by the operating system. The user must also provide an initial value for the PSW showing where execution should begin within the operating system, and with what status (protection key and mask values). The initial PSW must be at location 0 of the user's operating system.

After assembling the program submitted, ASSIST-V prepares for a simulation run. The size of the simulated machine is determined to be the smallest multiple of 2K (2048) greater than or equal to the length of the program assembled. ASSIST-V then fills all uninitialized storage locations (e.g. areas declared by DS statements) with X'F7', fills all registers with X'F6', and assigns each 2K byte memory block a protection key of 0, with fetch protection applied.

Simulated Execution

To begin the simulation, ASSIST-V fetches the contents of location 0 and makes this the active value of the PSW. It then proceeds to fetch and interpret instructions, simulating their effects as described in [9]. In the event of a program exception (e.g. invalid opcode, protection violation, address out of range, etc.), the active PSW is stored at the program old PSW reserved location (X'28') and the contents of the program new PSW location (X'68') are made active. This action differs from that of regular ASSIST, in that ASSIST would normally terminate a user for such a violation, while ASSIST-V effects a program interruption. Under ASSIST-V execution proceeds using the PSW specified in the program interrupt new PSW.

ASSIST-V contains a table of the execution time for each machine instruction or I/O operation. After an instruction is performed or an I/O operation occurs, the contents of the interval timer (memory location X'50') are decremented accordingly. If the interval timer passes from zero to negative, an external interruption is signalled, and whenever the external interrupt bit (bit 7) of the PSW permits external interrupts, the storing of an old and fetching of a new PSW occurs.

The completion of an I/O activity, and also other conditions in I/O execution, signal I/O interruptions. The PSW swaps for these are controlled by the system mask, bits

0-6 of the PSW. The interruption mechanism is the same as described above for program and external interrupts.

Termination of a User Program

ASSIST-V continues to simulate instruction execution until one of three events occurs.

1. The user halts the simulation by executing an XOPC 25. (See the description of XOPC's in section III of the ASSIST-V User's Guide [5].)

2. The simulation halts due to exceeding the limit for real execution time, number of instructions executed, or amount of output produced.

3. The user enters an "interminable wait" state. An interminable wait is one in which

a. the CPU is in wait state (bit 14 of the active PSW is '1'), and

b. no external interrupt can occur "reasonably" soon (the interval timer contains a negative value or the external interrupt mask in the PSW contains a '0'), and

c. no I/O interruptions can occur (either no channel-level I/O is in progress, or the system mask bits for all active channels contain '0').

After the program terminates, ASSIST-V displays final statistics about the simulation. These include length of the simulation in timer units, percent of time the CPU and each channel were active, and percent of overlap by the processors.

If the simulation terminated abnormally (reason 2 or 3 above), or if the user requested a dump (XOPC 25), a termination dump is printed. This includes the final value of the PSW, a trace of the last 10 instructions, a trace of the last 10 branches and/or PSW swaps, and the final contents of the registers and memory of the simulated machine.

3. INTRODUCTION TO VIRTUAL MEMORY

In the early days of computers, memories were expensive and small. Programmers spent a large portion of their time on such systems trying to squeeze a program into memory, often using an algorithm that was slower, or otherwise less efficient than another, simply because the better algorithm was too big to fit into the computer memory. The traditional solution to this problem has been the use of secondary memory, such as tape, drum, and disk. Memory management techniques such as overlays and dynamic serial structures were developed to permit the execution of programs that were larger than the memory available to the user. One drawback to these techniques, however, was that the programmer was responsible for designing the memory management structures, arranging for the movement of pieces of program between secondary and main storage, and in general, managing the entire process without any help from the computer.

In the early 1960's a method of performing the overlay process automatically, without the knowledge of the programmer evolved; this method is now called virtual memory. Virtual memory separates the concepts of memory address space and memory locations. The address space, or the logical addresses are those to which a program can refer when a program is executed. Logical addresses are mapped to real address locations which are actual, hardwired physical memory locations. A memory map exists that maps logical addresses to real ones. Programs are written just as though there were enough main memory for the entire logical address space, even though that may not be the case. Programs may load from or store into any word in the logical address space, or branch to an instruction located anywhere within the logical address space without regard to the fact that there is actually not enough physical memory to contain the program. This process is completely transparent to the user. The user of such a system can write programs without realizing that such a system exists. The user can simply think that the computer has a big memory.

The only requirement within a computer system for virtual memory is the existence of a secondary memory large enough to contain the complete program. The logical address space is broken up into equal sized chunks called pages. Page size in different virtual memory systems ranges from 256 bytes to 4096 bytes per page, always being a power of

two. The physical address space is likewise broken into pieces, called page frames, so that each page frame of main memory is capable of holding exactly one page. When a reference is made to a logical address, the computer translates the logical address to a real one using page tables, which may be special hardware registers or reserved sections of memory. Under the IBM System/370 series of computers this process is referred to as the Dynamic Address Translation feature or "DAT".

4. VIRTUAL STORAGE IN THE SYSTEM/370

The System/370 Dynamic Address Translation feature uses both segmentation and paging to implement virtual memory. Segmentation is a memory management technique used by IBM operating systems to organize main storage. A programmer will code a program within a logical address space and the operating system will automatically segment it. The result is a program whose address space is one or more variable segments. Each segment is further subdivided into one or more pages that are fixed in size. Page size in the System/370 can be fixed at 2048 or 4096 bytes per page. The logical address space is multilevel, structured into segments, pages within segments, and locations within pages. The logical address space is referenced by dividing the address into corresponding parts. Figure 1 shows how the address is structured.

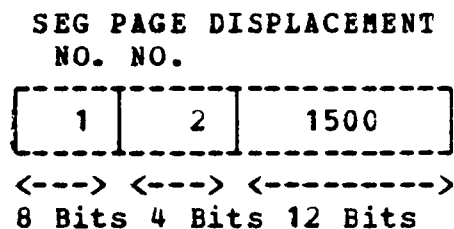


Figure 1. IBM address structure.

The first 8 bits of the address contain the segment number. This allows up to 256 segments in a program address space. The next 4 bits make up the page number allowing for 16 pages per segment. The displacement portion of the address consists of 12 bits which indicates, within the page, the location of the data being addressed relative to the first location within the page. A 12 bit displacement size allows a page size of 4096 bytes. The above three part address structure is able to account for all of IBM's 16,777,216 byte logical address space; 256 segments, each containing a maximum of 16 pages of size 4096 bytes gives a total maximum address of 16,777,216 bytes.

A page may be placed into any available page frame in main memory. The real address of where the page is placed in main memory is recorded in the page table. Each segment of a program has a page table. When a reference is made to a logical memory address, the segment number portion of the address is used as a pointer into the segment table, which contains the address of the page table for the current segment. The page number portion of the address becomes an index into the page table to the position where the real

storage location of the referenced page has been placed in memory. The Dynamic Address Translation feature then concatenates the page frame address and the displacement together to form an absolute address. Figure 2 shows the mapping of a logical address to real.

Before a program begins execution, the operating system loads Control Register 1 with the address of the program's segment table. In Figure 2, a reference has been made to logical address (hexadecimal) 013F24. The segment number in the upper 8 bits of the address is used as an index into the segment table. The segment table entry points to the beginning of the page table for the currently referenced segment. The page number from bits 8-11 of the address is used as an index into the page table at storage location 30000. The entry at this location identifies the beginning address of the page frame which contains the referenced address. The displacement portion of the logical address is then concatenated with the address of the page frame to give the real storage location address of 128F24.

As previously mentioned, virtual storage is typically larger than real storage, being bounded only by the number of bits which specify the address of a memory location and not the number of real storage locations available. The advantages of a virtual memory system are enhanced through its capability to "automatically overlay" programs in page size increments, requiring only that the immediately needed

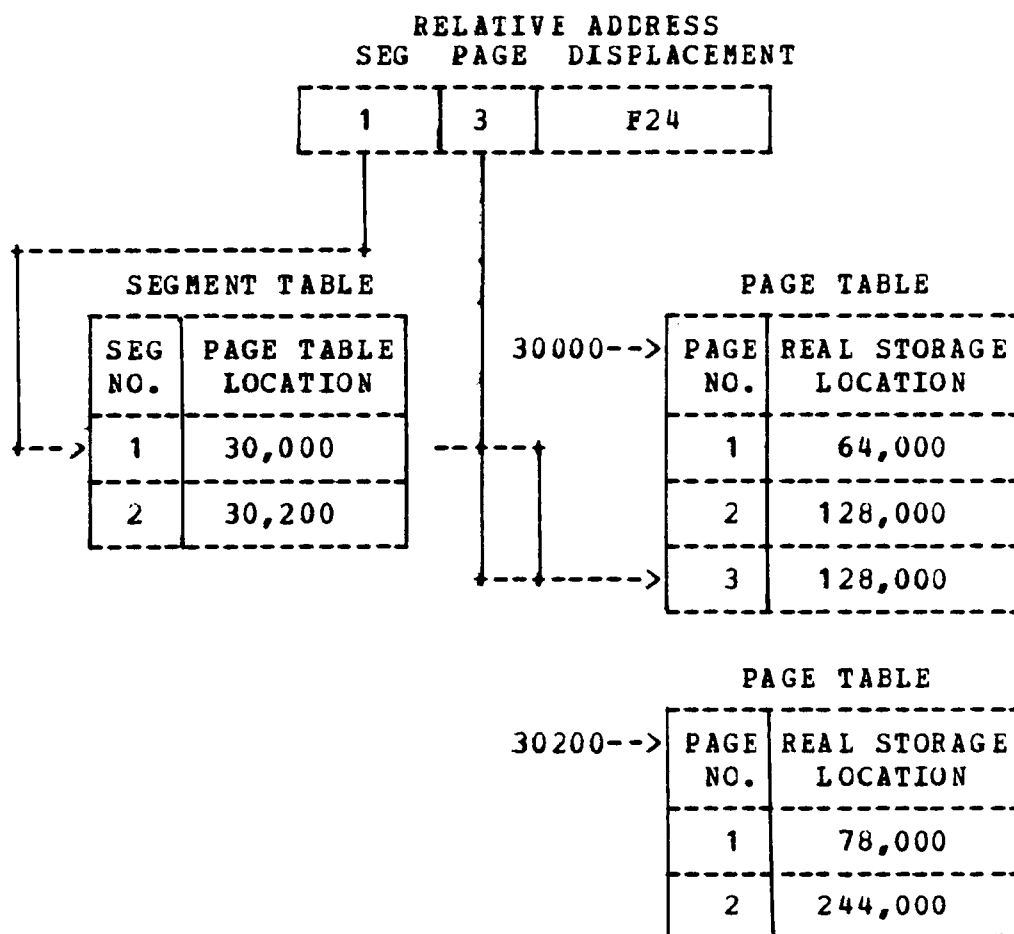


Figure 2. Address translation.

pages be in real storage at any one time during execution. This is true even if the entire program could fit into real storage. If a reference is made to a page which is not resident in real storage, the referenced page can be read into real storage from external storage and execution resumed. This process is known as "demand paging".

In a system where virtual storage is larger than real storage, all of the available page frames in real storage will eventually be filled. A reference to a page in this situation requires that a page residing in real storage be replaced to make room for the referenced page to be brought in. In such a case, any pages residing in a page frame allocated to users may be chosen for replacement. There are several page replacement strategies used to choose which page in real storage is to be replaced. The reader is referred to Madnick and Donovan [11] for a discussion of page replacement algorithms as well as a discussion of the interrupt action associated with page faults.

The System/370 uses a least recently used page replacement algorithm to manage real storage. In this page replacement strategy, the page frame which has not been referenced for the longest period of time is chosen for replacement. A page out operation is necessary only if the page chosen for replacement has been changed since it was paged in. This requires that a record of individual page usage be kept to assist the operating system in determining

which page to remove, and if a paging-out is required. This has been implemented in the System/370 by extending the key in storage by two additional bits. Bit 5, the reference bit, normally is set to one each time a location in the corresponding storage block is referred to for storing or fetching of information. Bit 6, the change bit, is set to one each time information is stored in the corresponding storage block.

5. USING THE VIRTUAL MEMORY OF THE EXTENDED INTERPRETER OF ASSIST-V

The user of ASSIST-V with virtual memory proceeds with preparing an "operating system" and a series of test programs for that system just as under standard ASSIST-V. However, with the virtual memory portion of ASSIST-V implemented, the user has the option of specifying a simulation of the System/370 with respect to the Dynamic Address Translation feature as well as a System/360 in the same fashion as specifying basic-control and extended-control modes of operation on a System/370. This section will describe the different operating procedures required for a simulation run when using the Dynamic Address Translation feature.

As with standard ASSIST-V the user's operating system is assembled and relocated as though the resultant object code were placed at location 0. However, if the test programs are to be executed in a virtual memory environment,

they too must be assembled relative to location 0, making each test program executable at any location in the simulated machine through dynamic address translation. Each test program is assembled separately under ASSIST-V, and the object module for that program is written either on an auxiliary storage device or on cards. The object decks for each test program are then written out by the user's operating system to an ASSIST-V virtual external storage device from which they may be paged in and out. The format of object decks produced by ASSIST-V is given in Figure 3.

When the DECK parameter is specified on the // EXEC card, an object deck is punched regardless as to whether the program was assembled with errors or not. It is the responsibility of the user to ensure the accuracy of the test programs before they are submitted to the user's operating system for execution. Unlike standard ASSIST-V, which will be terminated by an erroneous test program, the user's operating system will attempt execution of the test program if it has been assembled separately. The // EXEC parameters of NOLOAD and DECK are mutually exclusive so that the execution of the test programs may not be suppressed if an object deck is requested. Figure 4 shows the Job Control Language to assemble and punch object decks to an on-line disk for three test programs and then open the file for input, so the user's operating system can read the object decks in order to load them to an ASSIST-V virtual external storage device.

Two types of cards are punched and recognized: text cards (TXT), which supply actual object code, and end cards (END), which supply an optional entry point address for beginning of execution. The formats of these cards are described below. ALWAYS lists the characters which are definitely present, DECK notes those which are punched, and OBJIN those required for input. The notation IGNORED means that the given card columns are completely ignored when loading an object deck.

CARD/COLUMNS	ALWAYS	DECK	OBJIN
END CARD			
1		' '	IGNORED
2-4	END	-	-
5		X'00'	IGNORED
6-8		entry address	entry pt or blanks
9-72		blank	IGNORED
73-80		sequence no.	IGNORED
TEXT CARD			
1		' '	IGNORED
2-4	TXT	-	-
5		X'00'	IGNORED
6-8	beginning address of text code on this card		
9-10		blanks	IGNORED
11		X'00'	IGNORED
12	length of object code on card, from X'00' to X'38' (i.e. 0 to 56 decimal bytes of code).		
13-16		blanks	IGNORED
17-72	up to 56 bytes of code, to be loaded at given address.		
73-80		sequence	IGNORED

When ASSIST punches an object deck, it punches the entire program storage, including character 5's which fill any DS or other areas not having specified code values. Unlike the standard system assemblers, ASSIST always punches an END card with an entry point address on it, whether the user specifies an entry point on the source END card or not.

Figure 3. Object deck format.

```

//PROCLIB DD DSN=J30861.P39488.PROCLIB,DISP=SHR
// EXEC ASSIST,PARM='DECK'
//FT07F001 DD DSN=&TEMP,SPACE=(TRK,(2,1)),
// UNIT=SYSDA,DCB=(RECFM=FE,LRECL=80,
// BLKSIZE=3120),DISP=(NEW,PASS)
//SYSIN DD *
*** USER TEST PROGRAM
// EXEC ASSIST,PARM='DECK'
//FT07F001 DD DSN=&TEMP,DISP=(MOD,PASS)
//SYSIN DD *
*** USER TEST PROGRAM
// EXEC ASSIST,PARM='DECK'
//FT07F001 DD DSN=&TEMP,DISP=(MOD,PASS)
//SYSIN DD *
*** USER TEST PROGRAM
// EXEC ASSISTV,PROG=VIRTUAL
//STEPLIB DD DSN=J30861.P39488.ASSISTV.LOADLIB2,
// DISP=SHR
//SYSIN DD *
*** USER OPERATING SYSTEM
//INPUT DD DSN=&TEMP,DISP=(OLD,PASS)

```

Figure 4. Sample JCL for separate assemblies.

6. ASSIST-V ADDRESS TRANSLATION

Dynamic Address Translation provides the ability to interrupt the execution of a program at an arbitrary moment, record it on an external storage device, and at a later time return the program to any real storage location and resume its execution. Address Translation is accomplished by treating the addresses supplied by the program as logical addresses. These logical addresses are translated to real addresses through the use of translation tables created by the operating system. This section describes the support required by the user operating system to use the dynamic address translation facility of ASSIST-V.

Page size in ASSIST-V has been implemented in 256 byte blocks as opposed to System/370 page size of 2K or 4K byte pages. The choice to reduce the page size was based on two considerations; first, a small page size would be more practical in an academic environment, freeing the user from having to create large test programs, and second, a page size was needed that would fit completely on a single track of ASSIST-V virtual disk, which has a track capacity of 1692 bytes, which is less than the 2K page size of the System/370. To implement protection on such a system, main storage in ASSIST-V has been divided into storage blocks of 256 bytes, each with its own "key in storage" control field, rather than the standard 2K blocks.

The address structure is interpreted by the ASSIST-V Dynamic Address Translation feature as illustrated in Figure 5.

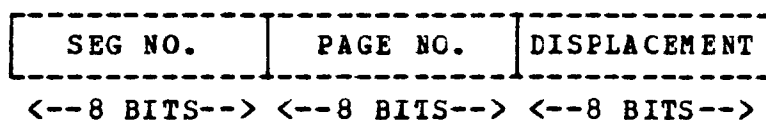


Figure 5. ASSIST-V address structure.

Bits 8-15 give the segment number, bits 16-23 give the page number, and bits 24-31 give the displacement within the page. When translation is performed, the segment and page numbers are used as indices into the segment and page tables respectively, and the logical displacement is concatenated with the page frame address in the page table to give the address of the real storage location.

With the dynamic address translation facility the CPU can operate either in translation mode or without address translation. If bit 12 of the PSW is a 1, the CPU will operate in extended mode, translating addresses, and if bit 12 is a 0, the CPU will operate in basic mode as a System/360. All other bits in the PSW are interpreted as they are in standard ASSIST-V.

Control registers provide a means for maintaining and manipulating control information that resides outside the PSW. ASSIST-V has been implemented with four control registers, which are registers 0-3. The instruction LOAD

CONTROL provides a means for loading control information from main storage into control registers; whereas, STORE CONTROL permits information to be transferred from control registers to main storage. These instructions operate in a manner similar to LOAD MULTIPLE and STORE MULTIPLE. Any attempt to reference a control register greater than 3 will result in a specification error.

Bits 0-28 of control register 1 designate the beginning and length of the segment table as illustrated in Figure 6.



Figure 6. Control register 1.

The fields in the register are allocated as follows: Bits 0-7 of control register 1 designate the length of the segment table. The length of the segment table is equal to one more than the value in bit positions 0-7. Bits 8-28 of control register 1 are a 24-bit real address that designates the beginning of the segment table.

The validity of the information loaded into control register 1 is not checked at the time the register is loaded. This information is checked and the program exception, if any, is indicated when the information is used.

Two types of translation tables are used for dynamic address translation - a segment table and a page table. These tables reside in main storage. The format of the tables is shown below.

Segment Table Entries

Each entry in the segment table designates the length and origin of the corresponding page table. An entry in the segment table consists of a full word. Figure 7 shows the format of a segment table entry.



Figure 7. Segment table entry.

Bits 0-7 designate the number of entries in the table. The length of the table is equal to one more than the value in bit positions 0-3. The length field is compared against the page-index portion of the logical address to determine if the page index is a legal entry in the page table.

Bits 8-28, with three low-order zeros appended, form a 24-bit real address that designates the beginning address of the page table.

Page Table Entries

The entry fetched from the page table indicates the availability of the page and contains the high order bits of

the real address. Each entry in the page table is three bytes in length. Figure 8 shows the format of a page table entry.



Figure 8. Page table entry.

Bits 0-15 provide the left-most 16 bits of a 24-bit real storage address. The page address and the contents of the displacement field of the logical address are concatenated to form a real address.

Bit 16 is the page-invalid bit. When the page-invalid bit is zero, translation proceeds using the table entry. When the page-invalid bit is one, a page-translation exception is recognized and a program interruption occurs. Bits 18-23 are unused.

7. TRANSLATION PROGRAM EXCEPTIONS

The Dynamic Address Translation feature recognizes two additional program exceptions:

Segment-Translation Exception

A segment-translation exception is recognized when:

1. The segment-table entry is outside the segment table.

Page-Translation Exception

A page-translation exception is recognized when:

1. The page-table entry is outside the page table.
2. The page-invalid bit has the value 1.

These program exceptions are recognized only as part of the execution of an instruction using address translation.

8. AN EXAMPLE

Figure 9 presents a sample operating system and three test programs executed by ASSIST-V with the Dynamic Address Translation feature implemented. Each test program is a separate CSECT assembled relative to location 0. The operating system consists of macro and DSECT definitions, an initialization section for the operating system and each test program, SVC, I/O, and program interrupt handlers, a loader to write the test programs to virtual disk, and internal I/O subroutines for reading in and writing out pages. The operating system is executed in basic-control mode without address translation while each test program is executed in extended-control mode with address translation. The test programs are executed in page frames with addresses different than those where each one was assembled. Each user is run under a separate protection key. The test programs terminate with an SVC which results in an OC2 privileged operation exception when executed by ASSIST; however, when the test programs are executed by ASSIST-V,

USER TEST PROGRAM 1

```

LOC  OBJECT CODE  ADDR1 ADDR2  STMT  SOURCE STATEMENT
000000 3 *****
4 *
5 *
6 *
7 *
8 *
9 *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *****

USER # 1 ==> CONVERTS EBCDIC TO BINARY
THIS PROGRAM DEMONSTRATES ASSIST-V-VIRTUAL'S CAPABILITY
TO:
1. EXECUTE INSTRUCTIONS SPLIT OVER PAGE BOUNDARIES. (LOC FE)
2. EXECUTE SS-TYPE INSTRUCTIONS WITH BOUNDARY VIOLATIONS
   ON OPERANDS.
3. EXECUTE INT-EDMK SS-TYPE INSTRUCTIONS WHICH MODIFY
   REGISTERS. (W/BOUNDARY VIOLATIONS)
4. EXECUTE THE "EXECUTE" INSTRUCTION IN A VIRTUAL
   ENVIRONMENT.
5. EXECUTE RX-TYPE INSTRUCTIONS OF LENGTH 8.(CVB)
  
```

```

000000 EQU 0
000001 EQU 1
000002 EQU 2
000003 EQU 3
000004 EQU 4
000005 EQU 5
000006 EQU 6
000007 EQU 7
000008 EQU 8
000009 EQU 9
000010 EQU 10
000011 EQU 11
000012 EQU 12
000013 EQU 13
000014 EQU 14
000015 EQU 15
000016 BALR R12,0
000017 USING *,R12
000018 L R0,=XL4'FF00FF00'
000019 LA R13,SAVEAREA
000020 LA R1,=A1ZONEVAL,50J
000021 BAL R14,CETNUM
000022 XDUMP 1
000023 SVC 1
000024
000025
000026
000027
000028
000029
000030
000031
000032
000033
000034
000035
000036
000037
000038
000039
000040
000041
000042
000043
000044 ZONEVAL DC 25CL1'
000045 CL5,=4096'
000046 DC 20CL1'
000047 SAVEAREA DS 18F
  
```

ESTABLISH ADDRESSABILITY

LOAD PTR TO SAVEAREA
 LOAD PTR TO ARGUMENTS
 CONVERT THE NUMBER
 SHOW RESULTS

Figure 9. User test program.

USER TEST PROGRAM 1

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT
49	***			50	SUBROUTINE GETNUM *** CONVERTS EBCDIC TO BINARY *****
51				52	THIS ROUTINE EXTRACTS A EBCDIC NUMBER FROM A FIELD (POSSIBLY) * PROCEEDED BY A SIGN), AND RETURNS THE BINARY FORM IN * REGISTER ZERO. THE NUMBER CAN OCCUR ANYWHERE IN THE FIELD. * INPUT ARGUMENTS ARE: * OIR1) -- ADDRESS OF FIELD CONTAINING THE NUMBER * 4IR1) -- LENGTH OF THE FIELD * EXIT CONDITIONS: * REGISTER 0 CONTAINS CONVERTED BINARY VALUE. * REGISTER 1 CONTAINS ADDRESS OF FIRST SPACE PAST * THE CONVERTED VALUE. * CONDITION CODE UPON EXIT: * 0 -- SUCCESSFUL CONVERSION * 2 -- INVALID NUMBER DETECTED * *****
67	0000C			68	GETNUM DS
69	0000C			70	LM R14,R12,12(R13)
71	00344			72	SH R3,R4,0(R1)
74	000B2			75	BNN SCAN
76	0000C			77	LM R14,R15,12(R13)
78	0000C			79	LM R2,R12,12(R13)
80	0010A			81	BR 14
82	00116			83	MVI R1,R1
84	00116			85	SR R2,R2
86	0010A			87	EX R4,SCANIRT
89	0010A			90	B'0110',BRANCHIRB-4(R2)
91	0010A			92	BAUFLO
93	0010A			94	PLUSIGN
95	0010A			96	NEUSIGN
97	00344			98	R5,R1
99	00344			100	R4,R3
101	00344			102	R4,R1
103	00344			104	BAUFLO
105	00344			106	R1,1(R4,R1)
107	00344			108	R4,SCANEND
109	00344			110	R1,R5
111	00344			112	R1,H'1'
113	00344			114	BAUFLO
115	00344			116	R1,PACNUM
117	00344			118	DMORD+7(1),SIGN
119	00344			120	AVN
121	00344			122	CVB
123	00344			124	SR R14,R14
125	00344			126	R1,1(R1,R5)
127	00344			128	EXIT

FORCE INS ACROSS PAGE BNDRY
 SAVE REGISTER CONTENTS
 LOAD ARGUMENTS INTO REGISTERS
 CHECK FOR LENGTH OF ZERO
 LENGTH IS OK
 SET CC = 2
 RESTORE REGISTER CONTENTS
 RESTORE REGISTER CONTENTS
 DEFAULT SIGN TO POSITIVE
 CLEAR R1 FOR TRI
 CLEAR R2 FOR TRI
 SCAN FOR START OF NUMBER
 NO NUMBER IN THE FIELD
 BRANCH ON NO NUMBER
 BRANCH FOR "+"
 BRANCH FOR "-"
 SAVE START OF NUMBER
 R4 = LEN CODE OF CHAR LEFT
 TO SCAN
 BR IF + OR - IN LAST BYTE
 R1 POINTS PAST LAST CHARACTER
 SCAN FOR CHARACTER PAST NUMBER
 R1 <= LENGTH OF NUMBER
 DECREMENT FOR PACK
 BR ON SIGN NOT FOLLOWED BY NUM
 PACK THE NUMBER
 ADD THE SIGN
 CONVERT INTO REGISTER 0
 SET CC = 0
 POINT R1 PAST LAST DIGIT

Figure 9. (Continued)

USER TEST PROGRAM 1

```

LOC  OBJECT CODE  ADDR1 ADDR2  STMT  SOURCE STATEMENT
CO0102  4110 1001      00001 105 PLUSIGN EQU *
CO0106  47FD COCC      0000E 106 LA R1,1(R1)
CO0106  47FD COCC      0000E 107 B PASTSIGN

CO010A 92DD C114      00116 109 NEGSIGN MVI SIGN,X'00'
CO010E 4111 0001      00001 110 LA R1,1(R1)
CO0112 47FD COCC      0000E 111 B PASTSIGN

CO0116  F270 C12E 5000 00130 00000 113 SIGN
CO0118  D030 303D C136 00000 00138 114 PACKNUM DS
CO011E  D030 303D C136 00000 00138 115 SCANTAT TRI DWORD,010,R51
CO0124  D000 5030 C236 00000 00238 116 SCANEND TRI 010,R51,TABEND
CO0130  0000000000000000 117 DWORD DS D
CO0138  0000000000000000 118 TABNUM DC XL78'00'
CO0186  04 0000000000000000 119 DC X'04'
CO0187  0070000000000000 120 DC XL17'00'
CO0198  08 0000000000000000 121 DC X'08'
CO0199  0000000000000000 122 DC XL143'00'
CO0228  0000000000000000 123 DC 10X'0C'
CO0232  0003C00000000000 124 DC XL6'00'

CO023A FFFFFFFF00000000 126 TABEND DC 240X'FF'
CO0328 0000000000000000 127 DC 10X'00'
CO0332  FFFFFFFF00000000 128 DC 6X'FF'
CO0338  0000001A00000032 129 END
CO0340  FFC0FF00 130
CO0344  0C01 131
CO0344  0C01 132

*** NO STATEMENTS FLAGGED - NO WARNINGS, 1936 HIGH, LEAVING 17456 FREE BYTES. AVERAGE: 72 BYTES/STMT ***
*** DYNAMIC CODE AREA USED: 7728 LOC +

```

```

*** CROSS-REFERENCE: VALUE(HEX) LOCATION REF REF ... (- SHOWS MODIFY) ***
RACFLD 0000A6 85 92 97 BRANCHTB 0000C6 84 DWORD 000130 -99 100 -114 EXIT 0000A8 103
GETINJ4 000096 40 NEGSIGN 00010A 87 PACKNUM 000118 98 PASTSIGN 0000CE 107 111 PLUSIGN 000102 86
R0 000000 -37 -100 R1 000001 -39 70 -74 74 -81 81 89 91 -93 93 -95 -96 98 -102
102 -106 106 -110 110 112 00000C -35 36 69 -77 R13 00000D -38 -69 76 77
R14 00000E -40 69 -76 -101 101 R15 00000F -76 R2 000002 -77 -82 82 84
R3 000003 -70 90 -115 R4 000004 -70 -71 83 -90 -91 93 94 R5 000005 -89
114 -116 SAVEAREA 00004C 38 SCAN 000082 72 SCANEND 000124 94 SCANTRT 00011E 83 SIGN
-80 99 -109 TABEND 000238 116 TABNUM 000138 115 ZONEVAL 00001A 130
*** ASSEMBLY TIME = 0.466 SECS, 285 STATEMENTS/SEC ***
*** PROGRAM EXECUTION BEGINNING - ANY OUTPUT BEFORE EXECUTION TIME MESSAGE IS PRODUCED BY USER PROGRAM ***

```

BEGIN XSNAP - CALL 1 AT C0000018 USER REGISTERS

```

REGS 0-7 00001000 00000038 00000012 00000000 F000FF00 00000338 F4F4F4F4 F4F4F4F4
REGS 8-15 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 80000012 F4F4F4F4 00000000

```

*** EXECUTION TIME = 0.016 SECS, 37 INSTRUCTIONS EXECUTED - 2312 INSTRUCTIONS/SEC ***
 *** FIRST CARD NOT READ: FILE UNOPENED

*** ASSIST COMPLETION DUMP ***

*** PSW AT ABEND FFC50002 4000001A COMPLETION CODE SYSTEM = 0C2 PRIVILEGED OPERATION

Figure 9. (Continued)

*** TRACE OF INSTRUCTIONS JUST BEFORE TERMINATION: PSW BITS SHOWN ARE THOSE BEFORE CORRESPONDING INSTRUCTION DECODED ***

IM LOCATION INSTRUCTION: IM = PSW BITS 32-39 (ILC,CC,MASK)

```

20 0000E 0100 C135 C114
EO 0000F4 4F00 C12E
AO 0000F8 18EE
40 0000FA 4111 5001
80 0000FE 47F0 C0A6
80 0000A8 98EF 000C
80 0000AC 982C 000C
80 0000C0 07FE
40 000012 E160 0000 0000
CO 000018 0A01

```

<-- LAST INSTRUCTION DONE - PROBABLE CAUSE OF TERMINATION

*** TRACE OF LAST 10 BRANCH INSTRUCTIONS EXECUTED ***

IM LOCATION INSTRUCTION: IM = PSW BITS 32-39 (ILC,CC,MASK)

```

00 00007E 45E0 C094
AO 0000A2 4780 C0B0
90 0000BE 4762 C1C0
90 0000C6 47F0 C100
90 000106 47F0 C1CC
80 0000FE 47F0 C0A6
80 000070 07FE

```

```

REGS 0-7 00001000 00000038 00000000 FF00FF00 00000338 F4F4F4F4 F4F4F4F4
REGS 8-15 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4

```

```

FLTR 0-6 F4F4F4F4F4F4F4F4 F4F4F4F4F4F4F4F4 F4F4F4F4F4F4F4F4 F4F4F4F4F4F4F4F4

```

*** USER STORAGE ***

```

C00030 05C05800 23E4100 C0A4110 C33035E0 C095E160 000000 000000 10 00038F 000000 40504040
C00020 404C4040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
000040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
000050 F000F000 00000338 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4
000080 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4 F4F4F4F4
0000A0 C342780 C0801F11 98EF003C 962C000C 07FE920F C1141811 18224440 C11C4762
0000C0 C0C047F0 C0A447F0 C13047F0 C1081851 1A431841 4740C0A4 41141001 4440C122
C000E0 1B154810 2342740 C0A44410 C116D110 C135C114 4F00C12E 18EE4111 500147F0
C00120 C0A44110 170147F0 C0CC9200 C1144111 000147F0 C0CC0FF5 F270C12E 50000000
C00120 3000C136 D0005J00 C236F5F5 F5F5F5F5 00000000 00000000 00000000 00000000 00000000
000140 00C00000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
LINES 000160-000160 SAME AS ABOVE
C00180 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0001A0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
LINES 0001C0-0001C0 SAME AS ABOVE
000220 00C00000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
C00240 FFF00000 FFF00000 FFF00000 FFF00000 FFF00000 FFF00000 FFF00000 FFF00000 FFF00000
LINES 000260-000260 SAME AS ABOVE
C00320 FFF00000 FFF00000 FFF00000 FFF00000 FFF00000 FFF00000 FFF00000 FFF00000 FFF00000
C00330 FFF00000 FFF00000 FFF00000 FFF00000 FFF00000 FFF00000 FFF00000 FFF00000 FFF00000
C00360 F5F5F5F5 F5F5F5F5 F5F5F5F5 F5F5F5F5 F5F5F5F5 F5F5F5F5 F5F5F5F5 F5F5F5F5 F5F5F5F5
LINES 000380-000380 SAME AS ABOVE

```

*** TOTAL RUN TIME UNDER ASSIST = 0.766 SECS ***

Figure 9. (Continued)

USER TEST PROGRAM 2

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT
2					*****
3					*****
4					*****
5					*****
6					*****
7					*****
8					*****
9					*****
10					*****
11					*****
12					*****
13					*****
14					*****
15					*****
16					*****
17					*****
18					*****
19					*****
20					*****
21					*****
22					*****
23					*****
24					*****
25					*****
26					*****
27					*****
28					*****
29					*****
30					*****
31					*****
32					*****
33					*****
34					*****
35					*****
36					*****
37					*****
38					*****
39					*****
40					*****
41					*****
42					*****
43					*****
44					*****
45					*****
46					*****
47					*****
48					*****
49					*****
50					*****
51					*****
52					*****
53					*****
54					*****
55					*****

C00000 0500
 C00002 00218
 C00006 00086
 C0000C 00000
 C00012 00043
 C00018 003AA
 C0001C 00084
 C00022 00143
 C00028 0016F
 C0002E 0016C
 C00034 00176
 C0003A 00173
 C00040 00131
 C00044 00131
 C0004A 00131
 C00050 00131
 C0005A 00131
 C00060 00131
 C00066 00131
 C0006C 00131
 C00072 00131

BALR 13,0
 USING *13
 LA 12,DATA
 XPRNT HEADNG,LEN1
 MVC ID1(80),0(12)
 CLC ID1(5),-9C*9
 BE DONE
 MVC ID1(80),ID
 MVC HOURS+3(12),HOURS+4
 PACK HRS(4),HOURS(5)
 MVC RATE+2(12),RATE+3
 PACK RT(4),RATE(4)
 CP HRS(4),-P*4000*
 BP OVERTIME
 ZAP PAY(9),HRS(4)
 MP PAY(9),RT(4)
 B OUTPUT
 SP PAY(10),HRS(4)
 SP PAY(10),-P*4000*
 MVN PAY(10),-P*150*
 AP PAY(9),-P*4000*
 MP PAY(9),RT(4)

LOAD BASE REGISTER
 TELL ASSEMBLER
 LOAD ADDRESS OF DATA AREA FOR INPUT
 PRINT LABELS AT TOP OF PAGE
 FETCH INPUT DATA
 CHECK FOR LAST CARD
 MAKE A COPY AT ID1(80)
 HOURS+4 COVER UP DECIMAL PLACE IN HOURS
 PACK HOURS WORKED
 COVER UP DECIMAL PLACE IN PAYRATE
 PACK THE PAYRATE
 SEE IF HOURS > 40
 IF SO TAKE INTO ACCOUNT AT OVERTIME
 PREPARE TO MULTIPLY HOURS BY RATE
 DO THE MULTIPLICATION
 OUTPUT THE RESULT
 MOVE HOURS TO MULTIPLY AREA
 COMPUTE OVERTIME HOURS
 TIMES TIME AND A HALF
 DIVIDE BY 100
 ADD THE 40 HOURS BACK IN
 MULTIPLY THE HOURLY RATE

COLS 1-9: EMPLOYEE NUMBER
 COLS 11-40: EMPLOYEE NAME
 COLS 42-47: HOURS WORKED
 COLS 49-52: HOURLY WAGES
 IN EACH NUMBER THERE MUST BE A DECIMAL PLACE WITH TWO DIGITS
 FOLLOWING IT AND IT MUST BE RIGHT-JUSTIFIED. A 40 HOUR
 WORK WEEK IS ASSUMED. SAMPLE DATA FOR THIS PROGRAM IS
 OBTAINED FROM WITHIN THE PROGRAM TO FURTHER DEMONSTRATE THE
 VIRTUAL MEMORY.
 THIS USER DEMONSTRATES ASSIST-V-VIRTUAL'S CAPABILITY
 TO:
 1. EXECUTE ALL CLASSES OF SS INSTRUCTIONS IN A VIRTUAL
 ENVIRONMENT WITH BOUNDARY VIOLATIONS, PRIMARILY ON
 OPERAND 1. (OPERAND FETCHED ACROSS 2 PAGES)

Figure 9. (Continued)

USER TEST PROGRAM 2

```

LOC OBJECT CODE ADDR1 ADDR2 STMT SOURCE STATEMENT
000078 F397 01C7 0133 57 UNPR PAYOUT(101,PAY(8) RESULT IN PAY, IGNORE LAST DIGIT
000079 D300 01D3 03B4 58 MVZ PAYOUT(911),=X*FO* CORRECT SIGN CODE
000080 D206 01C7 01C8 59 MVC PAYOUT(71),PAYOUT+1 MAKE ROOM FOR DECIMAL PLACE
000081 9218 01CE 00100 60 MVI PAYOUT+7,C*.* MOVE IN THE DECIMAL POINT
000082 4120 01C7 001C9 61 LA 2,PAYOUT REMOVE LEADING ZEROS
000083 95F0 2000 00000 62 NDIG CHECK FOR 0
000084 4770 00A6 000A6 63 BNE PRINT YES, HAVE FOUND FIRST NONZERO DIGIT
000085 9240 2000 00000 64 MVI 0121,C*.* REPLACE 0 WITH BLANK
000086 4122 0001 00001 65 LA 2,1121 INCREMENT ADDRESS REGISTER
000087 47F0 0090 00092 66 B NDIG LOOK AT NEXT CHARACTER
000088 EC20 0191 0085 00193 67 PRINT OUTPUT RESULTS
000089 41CC 0050 00050 68 LA 12,80112 INCREMENT TO NEXT INPUT ADDRESS
000090 47F0 000A 0000C 69 B NEXTPAY PROCESS ANOTHER EMPLOYEE
000091 0A02

```

```

000092 F0C5C4070306E8C5 72 HEADING DC
000093 4C404040404040 73 DC
000094 D5C1C4C5 74 DC
000095 4C404040404040 75 DC
000096 C8D6E4D9E240 76 DC
000097 D9C1E3C5404040 77 DC
000098 D7C1E840404040 78 DC
000099 80 HRS 79 LEN1 EQU
000100 81 PAY 80 HRS DS
000101 82 RT 81 PAY DS
000102 83 IU 82 RT DS
000103 84 NAME 83 IU DS
000104 85 HOURS 84 NAME DS
000105 86 RATE 85 HOURS DS
000106 87 OUTLINE 86 RATE DS
000107 88 IDEIC 87 OUTLINE DS
000108 89 PAYOUT 88 IDEIC DS
000109 90 69C*.* 89 PAYOUT DS
000110 91 LEN2 90 EQU
000111 92 DATA 91 LEN2 EQU
000112 93 92 DATA DC
000113 F4F9F1F5F0F4F0F7 94 DC
000114 F0F4F0F4F0F4F0F6 95 DC
000115 F4F1F0F9F0F3F0F6 96 DC
000116 F0F5F0F5F0F0F0F0 97 DC
000117 F0F4F0F8F1F4F7F6 98 DC
000118 F0F2F0F4F0F4F0F0 99 DC
000119 F0F2F0F8F1F4F7F9 100 DC
000120 F0F1F0F4F0F4F0F0 101 DC
000121 F0F1F0F4F0F4F0F0 102 DC
000122 F0F1F0F4F0F4F0F0 103 DC
000123 F0F1F0F4F0F4F0F0 104 DC
000124 F0F1F0F4F0F4F0F0 105 DC
000125 F0F1F0F4F0F4F0F0 106 DC

```

*** NO STATEMENTS FLAGGED - NO WARNINGS, NC ERRORS
 *** DYNAMIC CODE AREA USED 4184 LOW 1348 HIGH, LEAVING 178788 FREE BYTES. AVERAGE: 51 BYTES/STATEMENT ***

*** CROSS-REFERENCE: VALUE(HEX) LOCATION REF REF (-) SHOWS MODIFY) ***
 DATA 00C218 35 DONE 000084 39 HEADING 000086 36 HOURS
 -45 47 50 10 DONE 000143 -37 38 40 IDEIC 000194 -40 OUTPUT
 NOIC 000092 66 NEXTPAY 00000C 69 OUTLINE 000193 67 OUTPUT 000078 49 OVERTIME 000054 46
 000131 -42 HRS
 000085 67 LEN2
 000085 46 OVERTIME

Figure 9. (Continued)

[illegible]

EMPLOYEE NO.	NAME
491586073	TCIY PETERS
410903065	FRANCES PETERS
C40817763	DAPHNE PETERS
020819790	WHITNEY PETERS

HOURS	RATE	PAY
040.00	05.00	200.00
050.00	06.00	330.00
020.00	02.00	40.00
010.00	01.50	15.00

```

*** EXECUTION TIME = 0.033 SECS.          192 INSTRUCTIONS EXECUTED - 5810 INSTRUCTIONS/SEC ***
*** FIRST CARD NOT READ: NO CARDS READ:FILE UNOPENED

```

*** ASSIST COMPLETION DUMP ***

*** PSW AT ABEND FFC50002 40000086 COMPLETION CODE SYSTEM = OC2 PRIVILEGED OPERATION

*** TRACE OF INSTRUCTIONS JUST BEFORE TERMINATION: PSM BITS SHOWN ARE THOSE BEFORE CORRESPONDING INSTRUCTION DECODED ***

IM LOCATION	INSTRUCTION :	IM = PSW BITS 32-39 (ILC,CC,MASK)
00 000A2	47F0 D000	
80 00092	95F0 2000	
A0 00096	4770 D0A4	
A0 000A6	E020 D191 0085	
E0 000AC	41CC 0050	
A0 00090	47F0 D00A	
A0 0000C	D24F D141 C000	
E0 00012	0504 D141 D3A8	
C0 00038	4780 D0B2	
90 000084	0A02	

<-- LAST INSTRUCTION DONE - PROBLEM

*** TRACE OF LAST 10 BRANCH INSTRUCTIONS EXECUTED ***

IM LOCATION	INSTRUCTION : IM - PSW BITS 32-39 (ILC,CC,MASK)
20 00030	47F0 D00A
40 00050	47F0 D076
60 000A2	47F0 D090
80 000A2	47F0 D090
80 000A2	47F0 D090
80 C00A2	47F0 D090
80 000A2	47F0 D090
80 000A2	47F0 D070
A0 00096	4770 D0A4
A0 00080	47F0 D00A
80 00C10	4780 D0B2

RECS 0-7	F4F4F4F4	F4F4F4F4	00000000	00000000	00000000
RECS 8-15	F4F4F4F4	F4F4F4F4	F4F4F4F4	00000002	00000000
FLTR 0-6	F4F4F4F4	F4F4F4F4	F4F4F4F4	F4F4F4F4	F4F4F4F4

*** USER STORAGE ***

[illegible]

Figure 9. (Continued)

LINE	QTY	UNIT	PRICE	AMOUNT	TAX	TOTAL	DATE	TIME	NAME	STATUS	REMARKS
000000											
000001											
000002											
000003											
000004											
000005											
000006											
000007											
000008											
000009											
000010											
000011											
000012											
000013											
000014											
000015											
000016											
000017											
000018											
000019											
000020											
000021											
000022											
000023											
000024											
000025											
000026											
000027											
000028											
000029											
000030											
000031											
000032											
000033											
000034											
000035											
000036											
000037											
000038											
000039											
000040											
000041											
000042											
000043											
000044											
000045											
000046											
000047											
000048											
000049											
000050											
000051											
000052											
000053											
000054											
000055											
000056											
000057											
000058											
000059											
000060											
000061											
000062											
000063											

*** TOTAL RUN TIME UNDER ASSIST = 0.566 SECS ***

Figure 9. (Continued)

USER TEST PROGRAM 3

```

LOC OBJECT CODE ADDR1 ADDR2 SINT SOURCE STATEMENT
3 *
4 * USER # 3 ==> EXAMPLE USING CHARACTER MANIPULATION ACROSS PAGES*
5 *
6 * PROGRAM TO PRINT MAILING LIST ON LABEL, FOUR LABELS PER ROW.*
7 * ADDRESSES ARE OBTAINED FROM 77 BYTE AREAS WITHIN THE PROGRAM.*
8 *
9 * THIS PROGRAM DEMONSTRATES ASSIST-V-VIRTUAL'S CAPABILITY
10 *
11 * TO:
12 * 1. EXECUTE SS TYPE INSTRUCTIONS IN A VIRTUAL ENVIRONMENT
13 * WITH BOUNDARY VIOLATIONS ON BOTH OPERANDS, PRIMARILY
14 * OPERAND 1. (RESULT TO BE STORED ON SEPARATE PAGES)
15 *
*****

```

```

17 MAILLIST CSECT
18 BALR 13,0 LOAD BASE REGISTER
19 USING *1,3 TELL ASSEMBLER
20 LA 12,DATA LOAD ADDRESS OF DATA AREA FOR INPUT
21 LA 5,19 120 COLUMNS PER PRINTED LINES
22 LA 4,30 30 COLUMNS PER LABEL
23 RESTART POINTER TO CURRENT LABEL
24 MVC NAME(77),0(12) FEICH INPUT DATA
25 CLC NAME(5),-CL5***** IS THIS LAST INPUT RECORD?
26 BE DONE IF FOUND, QUIT PROCESSING
27 LA 6,LINE1(3) ADDRESS FOR NAME IN REGISTER &
28 MVC 0(27,6),NAME MOVE NAME FILLD TO 1ST OUT LINE
29 LA 6,LINE2(3) ADDRESS FOR STREET ADDRESS
30 MVC 0(25,6),STADDR MOVE STREET ADDRESS TO 2ND OUT LINE
31 LA 6,LINE3(3) ADDRESS FOR CITY-STATE OUT LINE
32 MVC 0(25,6),CITSTATE MOVE CITY-STATE TO 3RD OUT LINE
33 LA 12,77(12) INCREMENT TO NEXT RECORD
34 DXLE 3,4,NXTICARD PROCESS NEXT INPUT RECORD
35 XPRNT CCNT11,121 PRINT RESULTING & LABELS
36 XPRNT CCNT12,121 PRINT RESULTING & LABELS
37 XPRNT CCNT13,121 PRINT RESULTING & LABELS
38 B RESTART PROCESS NEXT ROW OF LABELS
39 EQU *
40 SVC 3

42 NAME DS CL27 INPUT AREA FOR NAME
43 STADDR DS CL25 INPUT AREA FOR STREET-ADDRESS
44 CITSTATE DS CL25 SPACE FOR INPUT CITY-STATE-ZIP
45 CCNT11 DC C'0' CARRIAGE CONTROL CHARACTER
46 LINE1 DC CL120' SPACE FOR FIRST OUTPUT LINE
47 CCNT12 DC C' ' CARRIAGE CONTROL CHARACTER
48 LINE2 DC CL120' SPACE FOR SECOND OUTPUT LINE
49 CCNT13 DC C' ' CARRIAGE CONTROL CHARACTER
50 LINE3 DC CL120' SPACE FOR THIRD OUTPUT LINE

```

Figure 9. (Continued)

USER TEST PROGRAM 3

LOC	OBJECT CODE	ADDR1	ADDR2	SIMI	SOURCE	STATEMENT
C00218	C099E5C905740E6			53	DC	CL27*IRVING WASHINGTON*
000233	F2F1F040E2D3C5			53	DC	CL25*2110 SLEEPY HOLLOW*
00024C	C4E4E3C3C8E30E6			54	DC	CL25*DUJCHTOWN, PA. 64479*
000265	D1C9D404C90540C5			55	DC	CL27*JIMMIN E. CRICKET*
000280	F2F2F240D7C9D5D5			56	DC	CL25*222 PINNOCHIO PLACE*
000299	D1E4D707C5E34C3			57	DC	CL25*PUPPET CITY, FLA. 72245*
0002B2	C5C4C7C1D94JC1D3			58	DC	CL27*EDGAR ALLEN POE*
0002C0	F4F740603C440C3			59	DC	CL25*47 OLD CROW DRIVE*
0002E6	E2C5C3C1E4C3E4E2			60	DC	CL25*SECAUCUS. N.J. 07090*
0002FF	C376D5D5C9C540C3			61	DC	CL27*CONNIE C. COMPUTER*
00031A	F9F0F4JC4E4D4D7			62	DC	CL25*906 DUMP DRIVE*
000333	C2C9E340C2E4C3D2			63	DC	CL25*81T BUCKET, KANSAS 41109*
00034C	5C5C5C5C5C5C4040			64	DC	CL77******
0003A0	5C5C5C5C5C5C			65	END	
				66		-CL5******

*** NO STALLMENTS FLAGGED - NO WARNINGS, 952 HIGH, LEAVING 180112 FREE BYTES. AVERAGE: 62 BYTES/STAT ***

*** DYNAMIC CORE AREA USED: 3256 LOW +

*** CROSS-REFERENCE: VALUE(HEX) LOCATION REF REF (SHOWS MODIFY) ***

CCN11	00004D	35	CCN12	000126	34	CCN13	00019F	37	CITSTATE	0J0094	32	DATA	000218	20
DCHE	00005E	26	LINE1	0000AE	27	LINE2	000127	29	LINE3	0001A0	31	NAME	000060	-24
ATCARD	070012	34	RESTART	00000E	38	STADDR	00007B	30					25	28

*** ASSEMBLY TIME = 0.233 SECS. 287 STATEMENTS/SEC ***

*** PROGRAM EXECUTION BEGINNING - ANY OUTPUT BEFORE EXECUTION TIME MESSAGE IS PRODUCED BY USER PROGRAM ***

IRVING WASHINGTON
2110 SLEEPY HOLLOW
DUJCHTOWN, PA. 64479

JIMMIN E. CRICKET
222 PINNOCHIO PLACE
PUPPET CITY, FLA. 72245

EDGAR ALLEN POE
47 OLD CROW DRIVE
SECAUCUS, N.J. 07090

CONNIE C. COMPUTER
906 DUMP DRIVE
81T BUCKET, KANSAS 41109

*** EXECUTION TIME = 0.016 SECS. 58 INSTRUCTIONS EXECUTED - 3625 INSTRUCTIONS/SEC ***

*** FIRST CARD NOT READ: NO CARDS READ: FILE UNOPENED

*** ASSIST COMPLETION DUMP ***

*** PSW AT ABEND FFC50002 40000060 COMPLETION CODE SYSTEM = 0C2 PRIVILEGED OPERATION

*** TRACE OF INSTRUCTIONS JUST BEFORE TERMINATION: PSW BITS SHOWN ARE THOSE BEFORE CORRESPONDING INSTRUCTION DECODED ***

IM LOCATION	INSTRUCTION: IM = PSW BITS 32-39 (ILC,CC,MASK)
20 000044	8734 D010
A0 07C048	E320 D0A8 0079
E0 00C04E	E320 D124 0079
E0 000054	E320 D19D 0079
E0 00005A	47F0 D00C
A0 00700E	4130 D000
A0 000012	D24C D05E C000
E0 000018	D504 D05E D39E
C0 00C01E	4780 D05C
80 00005E	0A03

<-- LAST INSTRUCTION DONE - PROBABLE CAUSE OF TERMINATION

*** TRACE OF LAST 10 BRANCH INSTRUCTIONS EXECUTED ***

IM LOCATION	INSTRUCTION: IM = PSW BITS 32-39 (ILC,CC,MASK)
20 00C044	8734 D010
A0 00C044	8734 D010
A0 000044	8734 D010

Figure 9. (Continued)

[illegible]

Figure 9. (Continued)

USE OF DYNAMIC ADDRESS TRANSLATION FEATURE

```

LOC  OBJECT CODE  ADDR1 ADDR2  STMT  *SOURCE STATEMENT
39                                     *SYSLIB EQUAREGS
40 LABEL  VIRTAB
41 .***>MACRO:VIRTAB *** GENERATES TABLES FOR PACING *****
42 .**
43 .** VIRTAB SETS UP THE SEGMENT AND PAGE TABLES FOR A USER
44 .** CN DOUBLE WORD BOUNDARIES. THE ADDRESS OF THE SEGMENT
45 .** TABLE IS PLACED ALONG WITH THE LENGTH IS PLACED INTO A FULL
46 .** WORD FOR LOADING INTO CONTROL REGISTER 1.
47 .** A LABEL IS THE ONLY REQUIRED PARAMETER.
48 .**
49 .*****
50 LABEL  DS  OD  DWORD ALIGN FOR SEG TABLE
51 STESYNDX DC  XL1*3*,AL3(PTESYNDX)  SEGMENT TABLE
52
53 PTESYNDX DC  XL3*00*  DWORD ALIGN FOR PAGE TABLE
54
55 DC  XL3*00*  PAGE TABLE ENTRY
56
57 DC  XL3*00*  PAGE TABLE ENTRY
58
59 DC  XL1*0*,AL3(STESYNDX)  FULL WORD ALIGNMENT
                                LENGTH AND 2 OF SEG TABLE
                                MEND

```

```

61 MACRO
62 LABEL  KEYSET CKEY=0,CRI=8,CR2=9,CADDR=NO
63 .***>MACRO:KEYSET *** SET STORAGE KEYS *****
64 .**
65 .** KEYSET SETS THE STORAGE KEY INDICATED IN THE PARAMETER CADDR
66 .** TO THE VALUE PASSED IN THE PARAMETER CKEY.
67 .**
68 .*****
69 AIF  (!CADDR EQ 'NO'),NOADDR
70 LABEL  LA  CRI,CKEY  LOAD USER KEY
71 SLL  CRI,4  SHIFT INTO BITS 24-28
72 L  CR2,XL4,CADDR  LOAD BLOCK ADDRESS
73 SSK  CRI,CR2  SET STORAGE KEY
74 MEXIT
75 .NOADDR  * *** NO ADDRESS GIVEN ***
76 MEND

```

Figure 9. (Continued)

USE OF DYNAMIC ADDRESS TRANSLATION FEATURE

```

LTC OBJECT CODE ADDR1 ADDR2 STMT SOURCE STATEMENT
78 MACRO
79 LABEL INSKY CR1=8,CR2=9,CADDR=NO
80 *==>MACRO:INSKEY *** INSERT STORAGE KEYS *****
81 *
82 * INSKY INSERTS THE STORAGE KEY IN THE PARAMETER CADDR
83 * INTO THE REGISTER PASSED IN THE PARAMETER CR1.
84 *
85 * *****
86 * AIF ('CADDR' EQ 'NO').NOADDR LOAD BLOCK ADDRESS
87 LABEL L CR2=XL4,CADDR* INSERT STORAGE KEY
88 ISK CR1,CR2
89 N CR1,=XL4'000000FF' CLEAR UPPER 3 BYTES
90 MEXIT
91 .NOADDR MNOTE * *** NO ADDRESS GIVEN ***
92 MEND

```

```

94 *==>DSECT:VIRTDAT *****
95 *
96 * VIRTDAT DSECTS THE USER SEGMENT AND PAGE TABLE DATA. BOTH
97 * THE SEGMENT AND PAGE TABLES REQUIRE DOUBLE WORD BOUNDARY
98 * ALIGNMENT.
99 * CTLREG1 SUPPLIES THE LENGTH AND ADDRESS OF THE SEGMENT TABLE
100 * FOR LOADING INTO CONTROL REGISTER 1.
101 *
102 *****

```

C00000	104 VIRTDAT	DSECT	00	DOUBLE WORD ALIGNMENT ON SEGMENT TABLE
C00030	105 SEGTABLE	DS	0	SEGMENT TABLE
C00030	106	DS	00	DOUBLE WORD ALIGNMENT ON PAGE TABLE
C00008	107	DS	00	PAGE TABLE ENTRY
C00008	108 PAGETAB	DS	XL3	PAGE TABLE ENTRY
C00008	109	DS	XL3	PAGE TABLE ENTRY
C0000E	110	DS	XL3	PAGE TABLE ENTRY
C00011	111	DS	XL3	PAGE TABLE ENTRY
C00014	112	DS	OF	FULL WORD BOUNDARY ALIGNMENT FOR LOAD
C00014	113 CTLREG1	DS	F	LENGTH AND ADDRESS OF SEGMENT TABLE

Figure 9. (Continued)

USE OF DYNAMIC ADDRESS TRANSLATION FEATURE

```

LOC  OBJECT CODE  ADDR1 ADDR2  SIMT  SOURCE STATEMENT
C00000
115  TEST1      CSECT
116  *****
117  *
118  *
119  *
120  *
121  *
122  *
123  *
124  *
125  *
126  *
127  *
128  *
129  *
130  *
131  *
132  *
133  *
134  *
135  *
136  *
137  *
138  *
139  *
140  *
141  *
142  *
143  *
144  *
145  *****

THIS PROGRAM DEMONSTRATES ASSIST-V IN VIRTUAL ADDRESSING-
MODE. THE FOLLOWING ACTIONS OCCUR:
1. THE OBJECT CODE FOR EACH PREVIOUSLY ASSEMBLED USER IS
   WRITTEN OUT TO A SEPARATE TRACK OF CYLINDER 0 OF VIRTUAL
   DISK 101.
2. USER 1 IS READ INTO PAGE FRAMES FROM TRACK 0 OF CYLINDER
   0. THE PAGES ARE PLACED IN REVERSE ORDER INTO THE PAGE
   FRAMES. (I.E. PAGE 1 --> PAGE FRAME 4; PAGE 2 --> PAGE
   FRAME 3; ETC.). USER 1 PAGE TABLES ARE SET WITH THE
   PROPER TRANSLATION ADDRESSES.
3. THE USER PROTECTION KEY IS PLACED INTO THE PAGE FRAMES.
4. USER 1 EXECUTES TO COMPLETION.
5. USERS 2 AND 3 ARE READ IN REVERSE ORDER INTO ALTERNATE
   PAGE FRAMES. (I.E. USER 2 PAGE 1 --> PAGE FRAME 8;
   USER 1 PAGE 1 --> PAGE FRAME 7) ETC.). THE PAGE
   TABLES FOR USER 2 AND 3 ARE SET TO THE PROPER TRANS-
   LATION ADDRESSES.
6. USER 2 AND 3 PROTECTION KEYS ARE PLACED INTO THE PAGE
   FRAMES.
7. USER 2 EXECUTES TO COMPLETION.
8. USER 3 EXECUTES TO COMPLETION.
9. AFTER EXECUTION OF EACH USER, THE STORAGE KEYS ARE
   DUMPED SHOWING THE REFERENCE AND CHANGE BITS FOR THAT
   USER.
10. MAIN STORAGE IS DUMPED WHEN ALL PROGRAMS HAVE COMPLETED
    EXECUTION.

```

```

C00000
C00001
C00002
C00003
C00004
C00005
C00006
C00007
C00008
C00009
C0000A
C0000B
C0000C
C0000D
C0000E
C0000F

EQU REGS
EQU 0
EQU 1
EQU 2
EQU 3
EQU 4
EQU 5
EQU 6
EQU 7
EQU 8
EQU 9
EQU 10
EQU 11
EQU 12
EQU 13
EQU 14
EQU 15

```

Figure 9. (Continued)

PAGE 4

USE OF DYNAMIC ADDRESS TRANSLATION FEATURE

LDC	OBJECT CODE	ADDR1 ADDR2	STMT	SOURCE	STATEMENT	COMMENT
C00030			165		USING *0,1	
C00033			166	IPLPSW	DC	DECLARE DSECT BASE REGISTER
C00000	00000000000000000080		167		USING VIRTUAT,R10	BEGIN AT INITIAL-->BC MODE
C00038			168	EXTOLD	DS 20	UNUSED
C00018			169	EXTOLD	DS 0	FOR EXTERNAL OLD PSW
C00020			170	SVCOLD	DS 0	FOR SUPERVISOR OLD PSW
C00028			171	PGHOLD	DS 0	FOR PROGRAM OLD PSW
C00030			172		DS 0	UNUSED
C00038			173	IGOLD	DS 0	FOR I/O OLD PSW
C00040			174	CSW	DS 0	CHANNEL STATUS WORD
C00048			175	CSW	DS F	CHANNEL ADDRESS WORD
C0004C			176		DS F	UNUSED
C00050			177	TIMER	DS F	INTERVAL TIMER
C00054			178		DS F	UNUSED
C00058			179		DS D	EXTERNAL NEW PSW
000060	4C00CC000000002A0		180	SVCNEW	DC	SUPERVISOR NEW PSW
000068	40000000000000352		181	PGNEW	DC	PROGRAM NEW PSW
C00070			182		DS D	UNUSED
000078	0000C000000000354		183	IGNEW	DC	I/O NEW PSW

Figure 9. (Continued)

USE OF DYNAMIC ADDRESS TRANSLATION FEATURE

LC	OBJECT CODE	ADDR1	ADDR2	SIMT	SOURCE INITIAL	STATEMENT DS	OH	
C00080	9401 0360		00360	185	LM	RO,R1,-A(1TEST1,HICORE)		GET BOUNDS FOR TRACE
C00080	5820 0370		00370	186	L	R2,-XL4*00008000*	3	TRACE PSW SHAPS
C00080	0103			187	XOPC			ENABLE TRACING
C00080	9401 0368		00368	188	LM	RO,R1,-A(1TEST1,TEST1*4096)		LOAD BASE REGISTERS
C00080	4500 0300		00800	189	BAL	R14,LOADER		LOAD USERS TO DISK
				190		URING USER 1 INTO CORE		
				191	LA	R10,USER1		LOAD DSECT BASE REGISTER
				192	LA	R3,0		LOAD TRACK NUMBER
				193	STC	R3,INRC+5		SHOW TRACK NUMBER
				194	LA	R7,XX08BUF4		GET 4 OF XX08BUF4 FRAME 1
				195	LA	R0,PAGETAB		LOAD INDEX FOR PAGE TABLE
				196	LM	3,5,-F*1,1,1,4*		LIMIT UN PAGES
				197	ST	R7,TEMP		SET UP 4 FOR MVC TO CCM
				198	MVC	READ(1,13),TEMP+1		MOVE 4 INTO CCM
				199	MVC	O(2,R8),TEMP+1		SHOW PAGE FRAME 2 IN PAGE TAB
				200	STC	R3,INRC+6		SHOW PAGE # -> TRK=0
				201	BAL	R14,FETCH		BRING IN A PAGE
				202	LA	R8,3(R8)		INCREMENT TO NEXT PAGE TAB ENT
				203	S	R7,-F*256*		DECREMENT TO NEXT PAGE
				204	BXLE	3,4,INLOOP1		LOOP FOR ALL PAGES
				205				
				207		LOAD CONTROL REGISTER		
				209	LCCTL	1,1,CTLREG1		LOAD CONTROL REGS
				210	STCTL	1,1,CTLCHK		DEBUG CHECK VALUES
				212		SET PROTECTION KEYS FOR THIS USER		
				214	KEYSET	KEY=3,R1=8,R2=9,ADDR=1000		
				215	LA	8,3		LOAD USER KEY
				216	SLL	8,4		SHIFT INTO BITS 24-28
				217	L	9,-XL4*1000*		LOAD BLOCK ADDRESS
				218	SSK	8,9		SET STORAGE KEY
				219	KEYSET	KEY=3,R1=8,R2=9,ADDR=1100		
				220	LA	8,3		LOAD USER KEY
				221	SLL	8,4		SHIFT INTO BITS 24-28
				222	L	9,-XL4*1100*		LOAD BLOCK ADDRESS
				223	SSK	8,9		SET STORAGE KEY
				224	KEYSET	KEY=3,R1=8,R2=9,ADDR=1200		
				225	LA	8,3		LOAD USER KEY
				226	SLL	8,4		SHIFT INTO BITS 24-28
				227	L	9,-XL4*1200*		LOAD BLOCK ADDRESS
				228	SSK	8,9		SET STORAGE KEY
				229	KEYSET	KEY=3,R1=8,R2=9,ADDR=1300		
				230	LA	8,3		LOAD USER KEY
				231	SLL	8,4		SHIFT INTO BITS 24-28
				232	L	9,-XL4*1300*		LOAD BLOCK ADDRESS
				233	SSK	8,9		SET STORAGE KEY
				235	XPRNT	-C*0*** USER 1 BEGINNING EXECUTION ***	35	
				236	LPSW	USERPSW		

Figure 9. (Continued)

USE OF DYNAMIC ADDRESS TRANSLATION FEATURE

LTC	OBJECT CODE	ADDR1	ADDR2	SYMT	SOURCE STATEMENT	BRING USER 2 INTO CORE	
				238 *	EXUSER2 EQU *		
000118	41A0 0258		00258	240	LA R10, USER2		LOAD OSECT BASE REGISTER
000118	41A0 0258		00258	240	LA R10, USER2		LOAD TRACK NUMBER
000118	4130 0001		00001	241	LA R3, 1		SHOW TRACK NUMBER
000120	4230 C090		00090	242	STC R3, INRC+5		GET 3 OF X00B0F8 FRAME 1
000124	4170 1700		01700	243	LA R7, X00B0F8		LOAD INDEX FOR PAGE TABLE
000128	4180 A0J8		00008	244	LA R8, PAGETAB		LIMIT ON PAGES
00012C	9835 0374		00374	245	LM 35, =F1, 1, 4		SET UP 3 FOR MVC TO CCM
000130	5C70 C29C		0029C	246	ST R7, TEMP		MOVE 3 INTO CCM
000134	0202 C2J9	029D 009D9	0029D	247	MVC READ+1(3), TEMP+1		SHOW PAGE FRAME 3 IN PAGE TAB
00013A	0201 80J0	029D 00000	0029D	248	MVC 012, (R8), TEMP+1		SHOW PAGE # -> TRK=1
000140	4230 C091		00091	249	STC R3, INRC+6		BRING IN A PAGE
000144	45E0 C8B6		008B6	250	BAL R14, FETCH		INCREMENT TO NEXT PAGE TAB ENT
000148	4138 0003		00003	251	LA R8, 3(R8)		DECREMENT TO SKIP NEXT PAGE
00014C	5870 C394		00394	252	S R7, =F5, 12		LOOP FOR ALL PAGES
000150	8734 0130		00130	253	BXLE 34, INLOOP2		
				255 *	LOAD CONTROL REGISTER		
000154	8711 A014		00014	257	LCTL 1, 1, CTLREG1		LOAD CONTROL REGS
000158	8611 028C		0028C	258	STCTL 1, 1, CTLCHECK		DEBUG CHECK VALUES
				260 *	SET PROTECTION KEYS FOR THIS USER		
00015C	4180 0002		00002	262	KEYSET KEY=2, R1=8, R2=9, ADDR=1100		LOAD USER KEY
000160	8580 0004		00004	263+	LA 8, 2		SHIFT INTO BITS 24-28
000164	5890 0388		00388	264+	SLL 8, 4		LOAD BLOCK ADDRESS
000168	0889		00088	265+	L 9, =XL4, 1100		SET STORAGE KEY
				266+	SSK 8, 9		
00016A	4180 0002		00002	267	KEYSET KEY=2, R1=8, R2=9, ADDR=1300		LOAD USER KEY
00016E	8580 0004		00004	268+	LA 8, 2		SHIFT INTO BITS 24-28
000172	5890 C390		00390	269+	SLL 8, 4		LOAD BLOCK ADDRESS
000176	0889		00089	270+	L 9, =XL4, 1300		SET STORAGE KEY
				271+	SSK 8, 9		
000178	4180 0002		00002	272	KEYSET KEY=2, R1=8, R2=9, ADDR=1500		LOAD USER KEY
00017C	8580 0004		00004	273+	LA 8, 2		SHIFT INTO BITS 24-28
000180	5890 C398		00398	274+	SLL 8, 4		LOAD BLOCK ADDRESS
000184	0889		00089	275+	L 9, =XL4, 1500		SET STORAGE KEY
				276+	SSK 8, 9		
000186	4180 0002		00002	277	KEYSET KEY=2, R1=8, R2=9, ADDR=1700		LOAD USER KEY
00018A	8580 0004		00004	278+	LA 8, 2		SHIFT INTO BITS 24-28
00018E	5890 C39C		0039C	279+	SLL 8, 4		LOAD BLOCK ADDRESS
000192	0889		00089	280+	L 9, =XL4, 1700		SET STORAGE KEY
				281+	SSK 8, 9		

Figure 9. (Continued)

USE OF DYNAMIC ADDRESS TRANSLATION FEATURE

LDC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT	
000194	41A0 0270	00270		283	BRING USER 3 INTO CORE	
000198	4130 0002	00302		284	LA R10, USER3	LOAD DSECT BASE REGISTER
00019C	4130 0002	00302		285	LA R3, 2	LOAD TRACK NUMBER
00019C	4230 0002	00302		286	STC	SHOW TRACK NUMBER
0001A0	4170 1600	01600		287	LA R7, XAOBBUFF	GET 2 OF XAOBBUFF FRAME 1
0001A4	4190 0008	00008		288	LA R8, PAGETAB	LOAD INDEX FOR PAGE TABLE
0001A8	9A35 0374	00374		289	LM 3,5=F,1,1,4	LIMIT ON PAGES
0001AC	5C73 029C	0029C		290	ST R7, TEMP	SET UP 2 FOR MVC TO CCM
0001B0	5232 0909	00290		291	MVC READ+1(3),TEMP+1	MOVE 2 INTO CCM
0001B6	5201 0000	00290		292	MVC 0(2,M8),TEMP+1	SHOW PAGE FRAME 2 IN PAGE TAB
0001BC	4230 0E91	00E91		293	STC R3, INRC+6	SHOW PAGE # -> TRK+3
0001C0	45E0 08B6	008B6		294	BAL R14, FETCH	BRING IN A PAGE
0001C4	4188 00J3	00303		295	LA R8, 3(1R8)	INCREMENT TO NEXT PAGE TAB ENT
0001C8	5870 0394	00394		296	S R7, F+512	DECREMENT TO NEXT PAGE
0001CC	8734 01AC	001AC		297	BXLE 3,4, INLOOP3	LOOP FOR ALL PAGES
				299 *	SET PROTECTION KEYS FOR THIS USER	
0001D0	4130 0001	00001		301	KEYSET KEY=L,R1=8,R2=9, ADDR=1000	
0001D4	8983 0004	00304		302+	LA R1, 8,1	LOAD USER KEY
0001D8	5870 0384	00384		303+	SLL R4, 8,4	SHIFT INTO BITS 24-28
0001DC	0489			304+	L 9,=XL4*1000	LOAD BLOCK ADDRESS
				305+	SSK R9, 8,9	SET STORAGE KEY
0001DE	4130 00J1	00001		306	KEYSET KEY=L,R1=8,R2=9, ADDR=1200	
0001E2	89A0 0004	00004		307+	LA R1, 8,1	LOAD USER KEY
0001E6	5890 038C	0038C		308+	SLL R4, 8,4	SHIFT INTO BITS 24-28
0001EA	0889			309+	L 9,=XL4*1200	LOAD BLOCK ADDRESS
				310+	SSK R9, 8,9	SET STORAGE KEY
0001EC	4180 0001	00001		311	KEYSET KEY=L,R1=8,R2=9, ADDR=1400	
0001F0	8780 0004	00304		312+	LA R1, 8,1	LOAD USER KEY
0001F4	5870 03A0	003A0		313+	SLL R4, 8,4	SHIFT INTO BITS 24-28
0001F8	0889			314+	L 9,=XL4*1400	LOAD BLOCK ADDRESS
				315+	SSK R9, 8,9	SET STORAGE KEY
0001FA	4180 00J1	00001		316	KEYSET KEY=L,R1=8,R2=9, ADDR=1600	
0001FE	8980 0004	00004		317+	LA R1, 8,1	LOAD USER KEY
000202	5870 03A4	003A4		318+	SLL R4, 8,4	SHIFT INTO BITS 24-28
000206	0889			319+	L 9,=XL4*1600	LOAD BLOCK ADDRESS
				320+	SSK R9, 8,9	SET STORAGE KEY
000208	EC20 0303	00303		322	XPRNT =C'0*** USER 2 BEGINNING EXECUTION ****,35	
00020E	8200 0230	00230		323	LPSM USER2PSM	
000212				325	EXUSER3	
000212	41A0 0270	00270		326	EQU *	LOAD DSECT BASE REGISTER
000216	8711 A014	00014		327	LA R10, USER3	LOAD CONTROL REGS
00021A	8611 028C	0028C		328	LCTL 1,1,CTLREG1	DEBUG CHECK VALUES
00021E	E020 03F6	003F6		329	STCTL 1,1,CTLCHECK	
000224	8200 0238	00238		330	XPRNT =C'1*** USER 3 BEGINNING EXECUTION ****,35	
					LPSM USER3PSM	

Figure 9. (Continued)

USE OF DYNAMIC ADDRESS TRANSLATION FEATURE

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT	
C00228	0039C000000000			332	USERPSW DC	00'0',X'00',B'00111001',XL2'0000',XL4'00000000'
				333	*	NO EXT INT,KEY=3-->ENABLED FOR EC MODE,ECMODE,PROB MOD
C00230	0029C000000000			334	USER2PSW DC	00'0',X'00',B'00101001',XL2'0000',XL4'00000000'
				335	*	NO EXT INT,KEY=2-->ENABLED FOR EC MODE,ECMODE,PROB MOD
C00238	0019C00000000000			336	USER3PSW DC	00'0',X'00',B'00011001',XL2'0000',XL4'00000000'
				337	*	NO EXT INT,KEY=1-->ENABLED FOR EC MODE,ECMODE,PROB MOD
000240				339	USER1	USER 1 SEG & PAGE TABLES
				340	USER1	DWORD ALIGN FOR SEG TABLE
000240	0300C248			341	ST0014	SEGMENT TABLE
000248				342	+	DWORD ALIGN FOR PAGE TABLE
000248	CC00C0			343	PT0014	PAGE TABLE ENTRY
000249	CC0000			344	+	PAGE TABLE ENTRY
00024E	CC00C0			345	+	PAGE TABLE ENTRY
000251	CC00C0			346	+	PAGE TABLE ENTRY
000254				347	+	FULL WORD ALIGNMENT
000254	CC00C240			348	+	LENGTH AND # OF SEG TABLE
000258				350	USER2	USER 2 SEG & PAGE TABLES
000258	0300C260			351	USER2	DWORD ALIGN FOR SEG TABLE
000260				352	ST0015	SEGMENT TABLE
000260				353	+	DWORD ALIGN FOR PAGE TABLE
000260	CC00C0			354	PT0015	PAGE TABLE ENTRY
000263	CC00C0			355	+	PAGE TABLE ENTRY
000266	CC00C0			356	+	PAGE TABLE ENTRY
000269	CC00C0			357	+	PAGE TABLE ENTRY
00026C				358	+	FULL WORD ALIGNMENT
00026C	CC00C258			359	+	LENGTH AND # OF SEG TABLE
000270				361	USER3	USER 3 SEG & PAGE TABLES
000270	0300C278			362	USER3	DWORD ALIGN FOR SEG TABLE
000278				363	ST0016	SEGMENT TABLE
000278	CC00C0			364	+	DWORD ALIGN FOR PAGE TABLE
000278	CC0000			365	PT0016	PAGE TABLE ENTRY
00027E	CC00C0			366	+	PAGE TABLE ENTRY
000281	CC00C0			367	+	PAGE TABLE ENTRY
000284				368	+	PAGE TABLE ENTRY
000284	CC00C270			369	+	FULL WORD ALIGNMENT
000288				370	+	LENGTH AND # OF SEG TABLE
000288				371	TABLEBUF DS	WORK BUFFER
00028C				373	CTLCHK DS	CTL REG DEBUG AREA
00029C				374	TEMP DS	WORK BUFFER

Figure 9. (Continued)

USE OF DYNAMIC ADDRESS TRANSLATION FEATURE

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT
0002A0	5810 03A8			376 *	*****
0002A0	1BFF			377 *	*****
0002A4	43F0 0023			378 *	SVC HANDLER
0002A6	43F0 0023			379 *	SHOW KEYS FOR USER STORAGE BLOCKS IN REGS. 4-7
0002A8	80F0 0002			380 *	AND BRANCH TO INITIATE EXECUTION OF NEXT USER.
0002AE	47FF 02AE			381 *	STORAGE IS DUMPED FOLLOWING TERMINATION OF LAST USER.
0002B2	47F0 02BE			382 *	*****
0002B6	47F0 02F0			383 *	*****
0002BA	47F0 0322				*****
0002BE					*****
0002C0	5890 0384		003A8	385	SVCHANDL DS OH
0002C2	0549		003B0	386	L R1,=A1TEST1+096J RESTORE BASE REGISTER
0002C4	5440 03AC		00023	387	SR 15,15 CLEAR R15 FOR INSERT CHAR
0002C8	5890 0388		00002	388	IC 15,SVCOLD+3 PICK OUT SVC# FROM OLD PSM
0002CC	0559		002AE	389	SLA 15,2 MUL BY 4 TO GET # OF SVC ROUTINE
0002CE	5450 03AC		002BE	390	B #+0(15) BRANCH TO PROPER ROUTINE
0002D0	0969		002F0	391	B SVC1CALL
0002D8	5460 03AC		00322	392	B SVC2CALL
0002DC	5890 038C		00384	393	B SVC3CALL
0002DE	0979		003AC	394	SVC1CALL ECU
0002E0	5470 03AC		00388	395	INSKEY R1=4,R2=9,ADDR=1000
0002E6	E160 0000	0000	0038C	396+	L 9,=XL4*1000*
0002EC	47F0 0118		003AC	397+	ISK 4,9
			00388	398+	N 4,=XL4*000000FF*
			003AC	399	INSKEY R1=5,R2=9,ADDR=1100
			0038C	400+	L 9,=XL4*1100*
			003AC	401+	ISK 5,9
			0038C	402+	N 5,=XL4*000000FF*
			003AC	403	INSKEY R1=6,R2=9,ADDR=1200
			0038C	404+	L 9,=XL4*1200*
			003AC	405+	ISK 6,9
			0038C	406+	N 6,=XL4*000000FF*
			00390	407	INSKEY R1=7,R2=9,ADDR=1300
			003AC	408+	L 9,=XL4*1300*
			00390	409+	ISK 7,9
			003AC	410+	N 7,=XL4*000000FF*
			00390	411	XDUMP
			003AC	412	B EXUSER2

Figure 9. (Continued)

USE OF DYNAMIC ADDRESS TRANSLATION FEATURE

LOC	OBJECT CODE	ADDR1	ADDR2	SMT	SOURCE CALL	STATEMENT ECU	
C002F0				414	SVC3CALL	INSKEY R1=4,R2=9,ADDR=1100	SHOW PROTECTION KEYS
C002F0	5890 0388		00388	415		L 9=XL4*1100*	LOAD BLOCK ADDRESS
C002F4 0549				416+		ISK 4,9	INSERT STORAGE KEY
C002F6 5440 03AC			003AC	417+		N 4=XL4*000000FF*	CLEAR UPPER 3 BYTES
C002FA 5890 0390			00390	418+		INSKEY R1=5,R2=9,ADDR=1300	SHOW PROTECTION KEYS
C002FF 0559				419		L 9=XL4*1300*	LOAD BLOCK ADDRESS
C00300 5450 03AC			003AC	420+		ISK 5,9	INSERT STORAGE KEY
C00304 5890 0398			00398	421+		N 5=XL4*000000FF*	CLEAR UPPER 3 BYTES
C00308 0959				422+		INSKEY R1=6,R2=9,ADDR=1500	SHOW PROTECTION KEYS
C0030A 5460 03AC			003AC	423		L 9=XL4*1500*	LOAD BLOCK ADDRESS
C0030E 5890 039C			0039C	424+		ISK 6,9	INSERT STORAGE KEY
C00312 0579				425+		N 6=XL4*030000FF*	CLEAR UPPER 3 BYTES
C00314 5470 03AC			003AC	426+		INSKEY R1=7,R2=9,ADDR=1700	SHOW PROTECTION KEYS
C00318 E160 0300			003AC	427		L 9=XL4*1700*	LOAD BLOCK ADDRESS
C0031E 47F0 0212			00212	428+		ISK 7,9	INSERT STORAGE KEY
C00322				429+		N 7=XL4*000000FF*	CLEAR UPPER 3 BYTES
C00322 5890 0384			00384	430+		XDUMP	
C00326 0549				431		B EXUSER3	
C00328 5440 03AC			003AC	432		SVC3CALL	
C0032C 5890 038C			0038C	433		INSKEY R1=4,R2=9,ADDR=1000	SHOW PROTECTION KEYS
C00330 0559				434		L 9=XL4*1000*	LOAD BLOCK ADDRESS
C00332 5450 03AC			003AC	435+		ISK 4,9	INSERT STORAGE KEY
C00336 5890 03A0			003A0	436+		N 4=XL4*000000FF*	CLEAR UPPER 3 BYTES
C0033A 0969				437+		INSKEY R1=5,R2=9,ADDR=1200	SHOW PROTECTION KEYS
C0033C 5460 03AC			003AC	438		L 9=XL4*1200*	LOAD BLOCK ADDRESS
C00340 5890 03A4			003A4	439+		ISK 5,9	INSERT STORAGE KEY
C00344 0579				440+		N 5=XL4*000000FF*	CLEAR UPPER 3 BYTES
C00346 5470 03AC			003AC	441+		INSKEY R1=6,R2=9,ADDR=1400	SHOW PROTECTION KEYS
C0034A E160 0000			00000	442		L 9=XL4*1400*	LOAD BLOCK ADDRESS
C00350 0119				443		ISK 6,9	INSERT STORAGE KEY
				444+		N 6=XL4*030000FF*	CLEAR UPPER 3 BYTES
				445+		INSKEY R1=7,R2=9,ADDR=1600	SHOW PROTECTION KEYS
				446		L 9=XL4*1600*	LOAD BLOCK ADDRESS
				447+		ISK 7,9	INSERT STORAGE KEY
				448+		N 7=XL4*000000FF*	CLEAR UPPER 3 BYTES
				449+		XDUMP	
				450		XOPC	
				451		25	ABEND WITH DUMP

Figure 9. (Continued)

USE OF DYNAMIC ADDRESS TRANSLATION FEATURE

```

LOC  OBJECT CODE  ADDR1 ADDR2  SIM1  SOURCE STATEMENT
453 *****
454 *
455 *          PROGRAM INTERRUPT HANDLER - ABEND WITH DUMP
456 *
457 *****

000352
000352 0119

459 PGMHANDL DS      OH
460          XOPC      25

462 *****
463 *
464 *          I/O INTERRUPT HANDLER
465 *          TAKE PROGRAM OUT OF WAIT STATE AND
466 *          TRANSFER TO NEXT INSTRUCTION
467 *
468 *****

000354 94FD 0039      00039
000358 8200 0038      00038

000360 0C07C0000000E92
000368 0C00C00000001000
000370 0C0C8000
000374 0C00000100000001
000380 0C00C100
000384 7C071000
000388 0CC01100
00039C 00001200
000390 00001300
000394 0000C200
000398 00001500
00039C 00001700
0003A0 00001400
0003A4 0C001600
0003A8 0CC01000
0003AC 0C0000FF
0003B0 F05C5C5C40E4E2C5
0003D3 F05C5C5C40E4E2C5
0003F6 F15C5C5C40E4E2C5

470 ICHANDL NI      IOOLD*1,8*11111101*      TURN OFF WAIT BIT
471          LPSW      IOOLD                      RESUME EXECUTION
472          LTRG

473          =A1TEST1,HICORE)
474          =A1TEST1,TEST1*4096)
475          =XL4*00008000*
476          =F*1,1,4*
477          =F*256*
478          =XL4*1000*
479          =XL4*1100*
480          =XL4*1200*
481          =XL4*1300*
482          =F*512*
483          =XL4*1500*
484          =XL4*1700*
485          =XL4*1400*
486          =XL4*1600*
487          =A1TEST1*4096)
488          =XL4*000000FF*
489          =C*0*** USER 1 BEGINNING EXECUTION ****
490          =C*0*** USER 2 BEGINNING EXECUTION ****
491          =C*1*** USER 3 BEGINNING EXECUTION ****

```

Figure 9. (Continued)

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT
				535	*** SUBROUTINE FETCH ***** READS A PAGE INTO MEMORY *****
				536	*****
				537	***** THIS ROUTINE READS A PAGE INTO MEMORY FROM VIRTUAL DISK. *****
				538	***** THE CALLER MUST PLACE THE PROPER CYLINDER, TRACK, AND RECORD *****
				539	***** NUMBERS INTO THE CONTROL AREA ASSOCIATED WITH THE CHANNEL *****
				540	***** COMMANDS. NO REGISTERS ARE MODIFIED. *****
				541	*****
				542	*****
C00P86				544	*****
				545	*****
C00P85	D203	0048	00048	546	*****
				547	*****
C0781C	E020	0020	00101	548	*****
				549	*****
C078C2	9C00	0101	00A20	550	*****
				551	*****
C008CA	E020	0020	0008D	552	*****
				553	*****
C008D0	07FE			554	*****
				555	*****
C00808	0000C9A8			556	*****
				557	*****
C0080C	F0D7C1C7C560C905			558	*****
				559	*****
C008ED	F0D7C1C7C560E2E4			560	*****
				561	*****
				562	*****
				563	*****

```

556 *** ==> SUBROUTINE PGOUT ***** WRITES A PAGE TO DISK FORM MEMORY *****
557 *
558 *
559 *
560 *
561 *
562 *
563 *

```

LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE STATEMENT
				565	*****
				566	*****
C0092A	D203	0048	00048	567	*****
				568	*****
C00910	E020	0020	0092C	569	*****
				570	*****
C00916	9C00	0101	00A20	571	*****
				572	*****
C0091A	8200	0A20	00A28	573	*****
				574	*****
C0091E	E020	0020	0093E	575	*****
				576	*****
C00924	07FE			577	*****
				578	*****
C00928	0000C9E0			579	*****
				580	*****
C0092C	F0D7C1C7C560D6E4			581	*****
				582	*****
C0093E	F0D7C1C7C560E2E4			583	*****
				584	*****

Figure 9. (Continued)

USE OF DYNAMIC ADDRESS TRANSLATION FEATURE

LDC	OBJECT CODE	ADDR1	ADDR2	SIMT	SOURCE	STATEMENT	OD	ALIGN	CCMS
C00960				577					
C00960	1F000E8440000001				PERFORM	DC			* CHANNEL PROGRAM 1 ==> POSITION HEAD & WRITE 5 RECORDS
C00960	07000E8440000006				DC				X'1F',AL31SFHASK,X'4000',H'1' SETFILE MASK
C00970	39000E8440000004				HGME	DC			X'07',AL31SEK,X'4000',H'6' SEEK CYL/HEAD
C00978	0800097040000000				DC				X'39',AL31SEK+2,X'4000',H'4' SEARCH HOME 2
C00980	15000A3040000000				DC				X'08',AL31HOMEL,X'4000',H'0' LOOP ON SEARCH
C00988	10000A4400000110				DC				X'15',AL31CCMR0,X'4000',H'20' WRITE RECORD 0
C00990	1D000B5440000110				DC				X'10',AL31CCMR1,X'4000',H'272' WRITE RECORD 1
C00998	1D000C6440000110				DC				X'1D',AL31CCMR2,X'4000',H'272' WRITE RECORD 2
C009A0	1D000C7420000110				DC				X'1D',AL31CCMR3,X'4000',H'272' WRITE RECORD 3
					DC				X'1D',AL31CCMR4,X'2000',H'272' WRITE RECORD 4
C009A8	1F000CE8440000001				PAGEIN	DC			* CHANNEL PROGRAM 2 ==> READ IN A PAGE
C009A8	07000E8440000006				DC				X'1F',AL31SFHASK,X'4000',H'1' SETFILE MASK
C009B8	39000E8440000004				HGME1	DC			X'07',AL31INRC,X'4000',H'6' SEEK CYL/HEAD
C009C0	3800058440000000				DC				X'39',AL31INRC+2,X'4000',H'4' SEARCH HOME 2
C009C8	31000E8440000005				ID	DC			X'08',AL31HOMEL,X'4000',H'0' LOOP ON SEARCH
C009D0	08000C4400000000				DC				X'31',AL31INRC+2,X'4000',H'5' SEARCH ID =
C009D8	06000C002000000100				READ	DC			X'08',AL31ID,X'4000',H'0' LOOP ON SEARCH
					DC				X'06',AL31+-X'2000',H'256' READ DATA
C009E0	1F000E84400000001				PAGEOUT	DC			* CHANNEL PROGRAM 3 ==> WRITE OUT A PAGE
C009E8	07000E8440000006				DC				X'1F',AL31SFHASK,X'4000',H'1' SETFILE MASK
C009F0	39000E8440000004				HGME2	DC			X'07',AL31INRC,X'4000',H'6' SEEK CYL/HEAD
C009F8	08000G6440000000				DC				X'39',AL31INRC+2,X'4000',H'4' SEARCH HOME 2
C00A00	31000E8440000005				ID1	DC			X'08',AL31HOMEL,X'4000',H'0' LOOP ON SEARCH
C00A08	08000A0040000000				DC				X'31',AL31INRC+2,X'4000',H'5' SEARCH ID =
C00A10	05000C002000000100				WRITE	DC			X'08',AL31ID,X'4000',H'0' LOOP ON SEARCH
					DC				X'05',AL31+-X'2000',H'256' WRITE DATA
C00A18	FF000C000000084E				WAIT1	DC			X'FF060000',A(RESTART) WAIT ON 1ST I/O PGM - BC MODE
C00A20	FF0600000000084E				WAIT2	DC			X'FF060000',A(RESTART) WAIT ON 2ND I/O PGM - BC MODE
C00A28	FF0600000000091E				WAIT3	DC			X'FF060000',A(RESTART) WAIT ON 3RD I/O PGM - BC MODE
C00A30	0C000C0000000004				CCWR0	DC			* DIRECT ACCESS RECORDS * * * * * KEY VALUE-----DATA-- * * * * *
C00A44	0C000C0000000000				CCWR1	DC			REC. NO. DATA LEN
C00B54	0C000C0000000000				CCWR2	DC			H'0',0',AL1(0,8),H'004',AL8'00000000000000',XL4'0000'
C00C64	0C000C0000000000				CCWR3	DC			H'0',0',AL1(1,8),H'256',AL8'00000000000000',XL256'FF'
C00D74	0C000C0000000000				CCWR4	DC			H'0',0',AL1(2,8),H'256',AL8'00000000000000',XL256'FF'
					CCWR4	DC			H'0',0',AL1(3,8),H'256',AL8'00000000000000',XL256'FF'
					CCWR4	DC			H'0',0',AL1(4,8),H'256',AL8'00000000000000',XL256'FF'
C00E84	CC				SFMSK	DC			8'11000000'
C00E85	0C000C0000000000				SEEK	DC			FILE MASK-ALLOW ANY SEEK, WRIT
C00E88	0C000C0000000000				INRC	DC			SEEK CYL, TRK 0
C00E92					MICORE	DC			CYL,TRK,REC,LEN, FOR PAGEIN

Figure 9. (Continued)

```

LOC  OBJECT CODE  ADDR1 ADDR2  STMT  SOURCE STATEMENT *****
626 *  *** -> SUBROUTINE GETPROG *** FETCHES OBJECT DECKS *****
627 *
628 *  THIS ROUTINE IS CALLED BY SUBROUTINE "LOADER" TO ASSEMBLE
629 *  AN OBJECT DECK INTO A HOLDING AREA (XXOBJUF1) TO BE WRITTEN
630 *  OUT TO VIRTUAL DISK.  EACH USER IS ASSEMBLED TO A RELATIVE
631 *  ADDRESS OF ZERO BY ASSIST.
632 *
633 *****
635 GETPROG DS OH
636 R4,R6,GPSAVE SAVE REGISTER CONTENTS
637 SR CLEAR FOR USE AS BYTE REGISTER
638 XARLOOP LA R4,XXOBJUF1 GET # OF HOLDING AREA FOR OBJ CODE
639 XARLX R4,XXOBJUF1 READ IN AN OBJECT DECK CARD
640 XARLX R4,XXOBJUF1 READ OF FILE?
641 CLC XXINBUFF+1(3),C*END* IS THIS END OF OBJECT CODE DECK?
642 BE XARLX YES, RETURN TO LOADER
643 IC R5,XXINBUFF+11 GET LENGTH OF CODE ON THIS CARD
644 BCTR R5,0 DECREMENT OF EX OF SS INSTRUCTION
645 L R6,XXINBUFF+4 GET BEGINNING # OF THE OBJ CODE
646 LA R4,0(R4,R6) GET # OF CODE IN HOLDING AREA
647 EX R5,MVCOBJCD EXECUTE MOVE TO HOLDING AREA
648 B XXRLCOP READ ANOTHER OBJECT CODE CARD
649 XARLX *
650 LM R4,R6,GPSAVE RESTORE REGISTER CONTENTS
651 BR R13 RETURN TO CALLER
653 MVCOBJCD MVC 0(R4,R6,XXINBUFF+16) MOVE OBJECT CODE TO HOLDING AREA
654
655 GPSAVE DS 6F REGISTER SAVE AREA
656 DS 6F ALIGNMENT OF INPUT BUFFER
657 XXINBUFF DS 80C INPUT BUFFER
658
659 MEMORY EQU * TEST1,X*1000*
660 XXOBJUF1 DC 256X11*AA* MEMORY PAGE FRAME
661 XXOBJUF2 DC 256X11*BB* MEMORY PAGE FRAME
662 XXOBJUF3 DC 256X11*CC* MEMORY PAGE FRAME
663 XXOBJUF4 DC 256X11*DD* MEMORY PAGE FRAME
664 XXOBJUF5 DC 256X11*EE* MEMORY PAGE FRAME
665 XXOBJUF6 DC 256X11*FF* MEMORY PAGE FRAME
666 XXOBJUF7 DC 256X11*99* MEMORY PAGE FRAME
667 XXOBJUF8 DC 256X11*88* MEMORY PAGE FRAME
668 LTRG
669
670 *-C*END*
END

```

Figure 9. (Continued)

```

PERF3PM 00C960 530 PGHARDL 00J352 181 PT0014 00J248 341 PT0015 00J260 352 PT0016 00J278 363 00J000 -186
READ 00J008 -199 -247 -291 RESTART1 00B8CA 609 RESTART2 0009IE 610 RO 00000D -507 651
R1 000001 -166 -189 -386 R10 00000A 166 -192 -240 -284 -326 R13 00000D -241 242 249 -285 286
R14 00C00E -190 -202 -250 -294 R2 000002 -187 R3 000003 -193 194 201 -241 242 249 -285 286
233 P4 000004 636 -658 -646 646 -650 -653 R5 000005 -637 637 -643 -644 647 R6 000006
636 -645 646 -650 R7 000007 -195 198 -204 -243 246 -252 -287 290 -296 R8 000008 -196 -200
233 203 -244 -248 -251 251 -268 -292 -295 295 SEEK 000E85 -520 581 582 SFMASK 000E84 580 591
670 ST0014 00J240 348 ST0015 00J258 359 ST0016 00J270 370 SVCHANDL 00J2A0 180 SVCOLD 00J020 388
SVCICALL 00J28E 391 SVC2CALL 00J2F0 392 SVC3CALL 00J322 393 TEMP 00J29C -198 199 200 -246 247 248 -290
291 292 TEST1 000000 473 474 474 487 USERPSW 00J228 230 USER1 00J240 192 USER2 00J258 240
LSER2PSW 00J230 323 USER3 00J270 284 326 USER3PSW 00J238 330 VIRTUAT 00J300 166 WAIT1 00J018 522
WAIT2 00J0A0 548 WAIT3 00J0A8 569 XXEDF 00J0C6 640 642 XXINBUFF 00J0EC -639 641 643 645 653
XXORRUF1 00J0U0 508 638 XXORRUF2 00J1C0 509 XXORRUF3 00J120 510 XXORRUF4 00J130 195 511 XXORRUF7 00J160 287
XXORRUF8 00J170 243 XXORRUF9 00J198 648
*** ASSEMBLY TIME = 2.233 SECS. 300 STATEMENTS/SEC ***
*** PROGRAM EXECUTION BEGINNING - ANY OUTPUT BEFORE EXECUTION TIME MESSAGE IS PRODUCED BY USER PROGRAM ***
TRACE--> TIME: 00J0084 PSM SWAP--CAUSE=I/O INT. CPSW= FF060101 0000084E; NPSW= 00000000 00000354
TRACE--> TIME: 00J18C4 PSM SWAP--CAUSE=I/O INT. OPSW= FF060101 0000084E; NPSW= 00000000 00000354
TRACE--> TIME: 00J26B2 PSM SWAP--CAUSE=I/O INT. OPSW= FF060101 0000084E; NPSW= 00000000 00000354
ALL JOBS ARE LOADED ON CYLINDER 0
PAGE-IN STARTING
TRACE--> TIME: 00J2F64 PSM SWAP--CAUSE=I/O INT. OPSW= FF060101 000008CA; NPSW= 00000000 00000354
PAGE SUCCESSFULLY BROUGHT IN
PAGE-IN STARTING
TRACE--> TIME: 00J3C6F PSM SWAP--CAUSE=I/O INT. OPSW= FF060101 000008CA; NPSW= 00000000 00000354
PAGE SUCCESSFULLY BROUGHT IN
PAGE-IN STARTING
TRACE--> TIME: 00J480E PSM SWAP--CAUSE=I/O INT. OPSW= FF060101 000008CA; NPSW= 00000000 00000354
PAGE SUCCESSFULLY BROUGHT IN
PAGE-IN STARTING
TRACE--> TIME: 00J53B1 PSM SWAP--CAUSE=I/O INT. OPSW= FF060101 000008CA; NPSW= 00000000 00000354
PAGE SUCCESSFULLY BROUGHT IN
*** USER 1 BEGINNING EXECUTION ***
BESTN XSNAP - CALL 1 AT 00J0018 USER REGISTERS
REGS 0-7 00J31300 00000038 80000012 F6F6F6F6 00000038 00038000 00000005
REGS 8-15 00J30001 00J30034 F6F6F6F6 00000F03 00030030 00030012 F6F6F6F6
TRACE--> TIME: 00J536E PSM SWAP--CAUSE=I/O INT. OPSW= 00J390001 4000001A; NPSW= 40000000 000002A0
BESTN XSNAP - CALL 2 AT 00J002EC USER REGISTERS
REGS 0-7 00001000 00001000 80C00012 F6F6F6F6 00000034 00030030 00000034
REGS 8-15 00J00001 00J01300 F6F6F6F6 00000F00 00000030 00030012 00000004

```

Figure 9. (Continued)

```

TRACE--> TIME: 0000586E PSM SNAP--CAUSE=I/O INT. OPSW= FF060101 000008CA; NPSW= 00000000 00000354
PAGE SUCCESSFULLY BROUGHT IN
PAGE-IN STARTING
TRACE--> TIME: 00006879 PSM SNAP--CAUSE=I/O INT. OPSW= FF060101 000008CA; NPSW= 00000000 00000354
PAGE SUCCESSFULLY BROUGHT IN
PAGE-IN STARTING
TRACE--> TIME: 00007468 PSM SNAP--CAUSE=I/O INT. OPSW= FF060101 000008CA; NPSW= 00000000 00000354
PAGE SUCCESSFULLY BROUGHT IN
PAGE-IN STARTING
TRACE--> TIME: 00007F36 PSM SNAP--CAUSE=I/O INT. OPSW= FF060101 000008CA; NPSW= 00000000 00000354
PAGE SUCCESSFULLY BROUGHT IN
PAGE-IN STARTING
TRACE--> TIME: 000086EA PSM SNAP--CAUSE=I/O INT. OPSW= FF060101 000008CA; NPSW= 00000000 00000354
PAGE SUCCESSFULLY BROUGHT IN
PAGE-IN STARTING
TRACE--> TIME: 000093F5 PSM SNAP--CAUSE=I/O INT. OPSW= FF060101 000008CA; NPSW= 00000000 00000354
PAGE SUCCESSFULLY BROUGHT IN
PAGE-IN STARTING
TRACE--> TIME: 00009FE4 PSM SNAP--CAUSE=I/O INT. OPSW= FF060101 000008CA; NPSW= 00000000 00000354
PAGE SUCCESSFULLY BROUGHT IN
PAGE-IN STARTING
TRACE--> TIME: 0000AAB7 PSM SNAP--CAUSE=I/O INT. OPSW= FF060101 000008CA; NPSW= 00000000 00000354
PAGE SUCCESSFULLY BROUGHT IN
*** USER 2 BEGINNING EXECUTION ***
EMPLOYEE NO. NAME HOURS RATE PAY
491586073 TONY PETERS 040.00 05.00 200.00
41093065 FRANCES PETERS 050.00 06.00 330.00
040819763 DAPHNE PETERS 020.00 02.00 40.00
020819790 WHITNEY PETERS 010.00 01.50 15.00
TRACE--> TIME: 0000AANC PSM SNAP--CAUSE=SVC INT. OPSW= 00290002 400000P4; NPSW= 40000000 000002A0
BEGIN XSNAP - CALL 3 AT 0000031E USER REGISTERS
REGS 0-7 00001000 00001908 0000015E 88888888 83888833 00030024 00000024
REGS 8-15 00000010 00001908 00000250 88888888 83888833 00030024 00000008

```

Figure 9. (Continued)

*** USER 3 BEGINNING EXECUTION ***

IRVING WASHINGTON
2110 SLEEPY HOLLOW
CUTCETOWN, PA. 64479
TRACE--> TIME: 0003ABIA PSW SNAP--CAUSE=SVC INT. OPSW= 00190003 40000060; NPSW= 40000000 000002A0

JIMMIN E. CRICKET
222 PINNOCHIO PLACE
PUPPET CITY, FLA- 72245

EDGAR ALLEN POE
47 OLD CROM DRIVE
SECAUCUS, N.J. 07090

CONNIE C. COMPUTER
906 DUMP DRIVE
BIT BUCKET, KANSAS 41109

BEGIN XSNAP - CALL 4 AT 00000350 USER REGISTERS

REGS 0-7	00001000	00001000	000001CE	00000000	00000016	00000016	00000016
REGS 8-15	00000010	00001600	00000270	00000F00	40000002	000001C4	0000000C

*** ASSIST-V FINAL STATISTICS ***

SIMULATED CLOCK TIME= 43805 TIMER UNITS
CPU UTILIZATION: BUSY 0.74 SUPER 0.52 PROBLEM 0.28
CHANNEL UTILIZATION: CH0 0.08 CH1 99.88 CH2 0.08
PROCESSOR OVERLAP: CPU/CHO 0.04 CPU/CH1 0.68 CPU/CH2 0.08
DISK ARM MOVEMENT: DISK101 15 SECS, 0 CYLINDERS CROSSED

*** EXECUTION TIME = 1.149 SECS. 1291 INSTRUCTIONS EXECUTED - 1123 INSTRUCTIONS/SEC ***

*** ASSIST COMPLETION DUMP ***

*** PSW AT ABEND 40000000 40000352 COMPLETION CODE ASSIST = V01 USER REQUESTED ABEND DUMP

*** TRACE OF INSTRUCTIONS JUST BEFORE TERMINATION: PSW BITS SHOWN ARE THOSE BEFORE CORRESPONDING INSTRUCTION DECODED ***

IM LOCATION INSTRUCTION: IM = PSW BITS 32-39 (ILC,CC,MASK)

90	000330	0959
----	--------	------

50	00C332	5450 03AC
----	--------	-----------

90	000336	5890 03A0
----	--------	-----------

90	00033A	0969
----	--------	------

50	00033C	5460 03AC
----	--------	-----------

90	00C340	5890 03A4
----	--------	-----------

90	000344	0979
----	--------	------

50	00C346	5470 03AC
----	--------	-----------

90	00034A	E160 0000 0000
----	--------	----------------

D0	00C350	0119
----	--------	------

<-- LAST INSTRUCTION DONE - PROBABLE CAUSE OF TERMINATION

*** TRACE OF LAST 10 BRANCH INSTRUCTIONS EXECUTED ***

IM LOCATION INSTRUCTION: IM = PSW BITS 32-39 (ILC,CC,MASK)

D0	00031E	47F0 0212
D0	00C224	8200 0238
A0	000044	8734 0010
A0	00C044	8734 0010
A0	000044	8734 0010
E0	00005A	47F0 000C
C0	00001E	4780 005C

Figure 9. (Continued)

the SVC instruction is recognized as legal and is executed accordingly.

The following describes the sequence of activities that take place as the test programs are executed by ASSIST-V with dynamic address translation.

1. A call is made to internal subroutine LOADER to write all three user test programs to ASSIST-V virtual disk. The loader, in turn, makes three successive calls to subroutine GETPROG which returns one complete user program at MEMORY(X'1000'). LOADER then writes out each program to virtual disk in page-size records with each program contained in a separate track. When LOADER returns, all test programs are stored on virtual disk.

2. The first user is loaded into four successive page frames, in reverse order. The address of the page frame into which each page is loaded is recorded in the page table as the page is written in.

3. Control register 1 is set to point to the beginning of the segment table.

4. User 1's protection key (key=3) is placed into each key in storage associated with the page frames allocated to user 1.

5. User 1 executes to completion, after which each key in storage is printed out in registers 4-7 showing the reference and change bits for this user.

6. Users 2 and 3 are separately loaded into memory

page frames in the same fashion as user 1. Users 2 and 3, however, are loaded into memory together with their pages in reverse order into alternating page frames; the first logical page of user 2 is loaded into the page frame at hexadecimal location 1700, while the first logical page of user 3 is loaded into the page frame at hexadecimal location 1600. As the pages are loaded into memory, the page tables for users 2 and 3 are set to reflect the real addresses of pages.

7. Control register 1 is loaded with the address of the segment table for user 2.

8. The protection keys for user 2 (key=2) and user 3 (key=1) are placed in each key in storage of each page frame allocated to users 2 and 3 respectively, before user 2 begins execution.

9. User 2 executes to completion after which each key in storage for the page frame allocated to user 2 is printed out in registers 4-7, showing the reference and change bits for user 3.

10. Control register 1 is loaded with the address of the segment table for user 3.

11. User 3 executes to completion after which the keys in storage are printed out in the same fashion as for users 1 and 2.

III. PROGRAM LOGIC MANUAL

1. PURPOSE

This document is intended for the person who will be creating and maintaining a version of ASSIST-V with Dynamic Address Translation. It contains information on how to assemble the Dynamic Address Translation feature into ASSIST-V, the technical changes made to ASSIST-V in implementing the Dynamic Address Translation feature, and suggestions for additions. A document you may wish to consult is ASSIST-V Program Logic Manual [4], which gives information on bringing up and maintaining ASSIST-V; the material in that document is assumed.

2. SYSGEN PARAMETERS

ASSIST-V with Dynamic Address Translation is an extension of ASSIST-V. A proper ASSIST-V module must be created for the Dynamic Address Translation feature to be included. Only two additional SYSGEN parameters must be set in ASSIST-V to include the Dynamic Address Translation feature; &VS370 and &S370A. The parameter &VS370 controls the assembly of the Dynamic Address Translation feature, while the latter parameter, &S370A, is required for the assembly of the System/370 instructions LOAD CONTROL and STORE CONTROL. When ASSIST-V is assembled to create a version with the Dynamic Address Translation feature, the data sets SOURCE1, SOURCE2, and SOURCE6 must be assembled.

SOURCE3 must be included if the present version of ASSIST-V does not permit the assembly of System/370 instructions.

3. DYNAMIC ADDRESS TRANSLATION OVERVIEW

The simulated execution-time Dynamic Address Translation feature consists primarily of translating logical addresses to real within ASSIST-V and routines to support the translation. Additionally, the routines which do address and protection checking have been changed along with the implementation of the keys in storage associated with each 2K block of memory. These changes were necessary to reflect a virtual memory with a page size of 256 bytes.

4. IN-LINE ADDITIONS TO ASSIST-V TO SUPPORT DAT

Addresses within ASSIST-V with the Dynamic Address Translation feature are treated as logical addresses which may or may not require translation. If the simulated machine is operating in extended control mode (bit 12 of the PSW equal to 1), address translation occurs and the operating system must supply the appropriate translation tables for the Dynamic Address Translation feature. If the simulated machine is running in basic control mode (bit 12 of the PSW equal to 0), no address translation occurs. The in-line changes to ASSIST-V consist primarily of placing the machine in the proper mode of execution (extended or basic control), translating logical addresses to real, and ensuring that all instructions and operands are contiguously available to ASSIST-V for simulated execution.

Before an instruction is fetched, the PSW is checked to see if the machine is running in basic or extended control mode. If the E-mode bit is set, bit X'40' of control ECFLAG4 is set. This bit is the flag controlling address translation within ASSIST-V. If the translation routine, DYNATRAN, is called and bit X'40' of ECFLAG4 is 1, the address is translated, otherwise the logical address is returned as real.

Prior to an instruction fetch, the address of the instruction is translated to obtain the real address of the next instruction. If the instruction lies entirely within a single page, the instruction is fetched just as under standard ASSIST-V; however, if the instruction is split across a page boundary, the separate parts of the instruction are concatenated in a 6-byte buffer, and the buffer address is substituted for that of the real address. This has no effect on ASSIST-V or on the user program. The updates to the instruction stack and the incrementing of the instruction address register are done with logical addresses. ASSIST-V is able to proceed normally with either its normal instruction buffer or the one substituted by DAT. If the instruction was an EXECUTE, the logical address of the out-of-line instruction is placed into the instruction stack by a change made in the simulation of the EXECUTE instruction.

The operands of an instruction are checked for page boundary violations just prior to being checked for addressing and protection violations. After the first byte of an instruction is decoded, the opcode, a multiway branch, is executed by ASSIST-V to decode the different configurations of the second byte of an instruction. ASSIST-V merges back after decoding the second byte with the logical address of the operand in register 1 and the operand length in register 0. At this point, the macro DYCKOP is invoked expanding into a call to internal subroutine DYPGVIO which determines if the operand is split across two pages. If so, a flag is set showing that the operand needs to be concatenated in a DAT buffer.

After the operand has been checked for addressing and protection exceptions, the operand is translated and, if split across a page boundary, concatenated into a DAT buffer with the address of the DAT buffer substituted for that of the operand address. If the instruction is an SS-type instruction, the address of the first operand is saved if a page violation occurs so the result of the instruction execution can be returned to the first operand.

Not all addressing protection checks are done in the main decoding loop by ASSIST-V. In the case of some RS-type instructions, the length of the operand is not given explicitly in the instruction and is not known until the instruction execution is simulated. Such is the case with

the instructions LM, STM, LCTC, and STCTL. In such cases, page violation checking and operand concatenation is done in the routines that simulate the instruction.

In most SS-type instructions, instructions that store the contents of multiple registers, and psuedo instructions that modify storage, it is possible that the first operand, the receiving operand, is split across two pages. In such cases, the DAT buffer address has already been substituted for the operand address and contains the result of the instruction execution. Control is deferred to a section of code at the label EICONCAT where, if a page violation exists, the result of the instruction execution is copied out to separate pages before returning to fetch another instruction.

5. SUPPORT ROUTINES FOR DAT

This section describes the out-of-line routines that simulate the Dynamic Address Translation feature. These routines are written as internal subroutines but establish their own addressability because of the lack of addressability remaining within the range of the base registers of CSECT EXECUTE.

Subroutine DYNATRAN

Subroutine DYNATRAN is the actual address translator. DYNATRAN receives a logical address in register 0 and returns a real one in register 0. If bit X'40' of ECFLAG4

is set to 0 (BC mode), the input parameter is returned as the real address. DYNATRAN checks for segment-translation, translation-specification, and page-translation exceptions in the process of following a chain of pointers beginning with control register 1 through the segment table to the page tables. If no errors are found, the real address of the page is concatenated with the displacement bits of the logical address to form a 24-bit real address. If a program exception is found, the address of the proper error routine is loaded into the return register.

Internal subroutine DYSETRC is called by DYNATRAN to set the reference and change bits in the keys of storage of the real address being referenced.

Subroutine DYINSCHK

Subroutine DYINSCHK checks to see if an instruction is split across a page boundary. If there are more than four bytes remaining on a page, all of the current instruction is on the current page; however, if not, there is a possibility that the instruction violates a page boundary. In such a case, DYINSCHK copies the remaining portion of the address into a DAT buffer, adds the length of that portion to the logical address and translates the result to locate the remaining part of the instruction. The rest of the instruction is then concatenated with the first part of the instruction in the buffer. The address of the buffer is substituted for the instruction address.

Subroutine DYPGVIO

Subroutine DYPGVIO checks an operand to see if the span of the operand is across a page boundary. Prior to a call to DYPGVIO, a translation flag is set by the caller indicating which operand is to be checked. This routine adds the length of the operand to its beginning address. If the result crosses a logical page boundary, the operand is not entirely within one page. If this is the case, the routine returns without resetting the flag showing which operand is being checked for a page violation. This flag is later used to determine if an operand is to be read into or written out from contiguous DAT buffers. If no violation is found, the flag is reset.

DYPGVIO assumes that the caller has requested for operand 2 to be checked. If this is not the case, operand 1 is checked. The determination of which operand is to be checked is order dependent. The caller specifies which operand is to be checked by setting one of two bits, each corresponding to an operand. The correct operand is chosen by inspection of these two bits. ASSIST-V checks operand 1 prior to operand 2, and if there is a violation on operand 1, operand 1's bit will remain set while operand 2 is being checked. If the initial assumption is made that a request has been made to check operand 1, operand 2 may never be checked. This problem is avoided by inspecting the bit specifying operand 2 before the bit specifying operand 1.

Subroutine DYMVOPER

Subroutine DYMVOPER concatenates an operand that is split across two separate pages into a contiguous DAT buffer. After a determination of which operand is to be copied by an inspection of bits set by the caller, the remaining data on the current logical page is copied into a buffer. The remaining portion of the operand is located by translating the sum of the logical address of the operand and the remaining number of bytes on the current page. The buffer address is substituted for that of the operand by the caller.

Subroutine DYMVOPIN

Subroutine DYMVOPIN is the analog of DYMVOPER, writing out an operand from a contiguous DAT buffer to two separate pages. The method of operation is similar to that of DYMVOPER, except in the direction of the data movement.

6. IMPLEMENTATION OF CONTROL REGISTERS

In the System/370 control registers provide a method of maintaining and manipulating information that resides outside the PSW. Control registers 0 and 1 are used by the System/370 during dynamic address translation. Control register 0 specifies the page size in the System/370 and is implemented in ASSIST-V but unused. Control register 1 specifies the length and address of the segment table and is simulated by ASSIST-V. Control registers 2 and 3 have been

reserved for future use. Additional control registers can be implemented by adding the additional storage for the fake control registers ECONREGS in DSECT ECCNTROL.

Information in control registers is fetched and stored by the instructions LOAD CONTROL and STORE CONTROL. These instructions have been added to the instruction set of ASSIST-V. LOAD CONTROL and STORE CONTROL are described in the IBM System/370 Principles of Operation [10]. Any attempt to reference a control register not currently implemented by ASSIST-V results in an OC6 specification exception.

Until all control registers are implemented, wraparound is not permitted and results in an OC6 specification exception. If and when a full set of control registers is installed, this prohibition can be removed by deleting lines 85612220-85612240 in the simulation of LOAD CONTROL and STORE CONTROL. The code to support wraparound has been added to ASSIST-V for the time when a full set of control registers is present.

LOAD CONTROL and STORE CONTROL are executed by the same code which simulates LOAD MULTIPLE and STORE MULTIPLE after some preliminary error checking is done and a flag is set to indicate LOAD CONTROL and STORE CONTROL are being executed. Some flags have been added to the simulation which enable the instruction simulation to substitute the address of the fake control registers in place of the fake general purpose registers.

7. CHANGES TO ASSIST-V FOR DAT

This section describes the changes made to standard ASSIST-V in simulating the Dynamic Address Translation feature.

The primary and only major change to ASSIST-V is how main storage is divided into blocks. When the Dynamic Address Translation feature is assembled into ASSIST-V, main storage is divided into 256-byte blocks for purposes of protection and recording reference and change information. This change was necessary so individual page frames in real storage could be protected and reference and change information for each page could be kept in bits 5 and 6 of the keys in storage.

The instructions INSERT STORAGE KEY and SET STORAGE KEY have been changed to reflect the new block size of 256 bytes. INSERT STORAGE KEY and SET STORAGE KEY are executed as described in IBM System/370 Principles of Operation [10] except as with the above interpretation of the keys in storage and one further change; when the PSW specifies basic-control mode, the complete 7-bit key is placed into bit positions 24-30 of the register specified in the R1 field. If this were not the case, the user operating system would not be able to inspect the reference and change bits when the user operating system was executing in basic-control mode.

The change in the way ASSIST-V interprets the keys in storage has required a change to be made to the routine that

checks for protection exceptions. Standard ASSIST-V simply checked contiguous blocks of storage, beginning with the highest referenced block of storage through the lowest block. Since blocks (or pages) may not be contiguous when a program is executing in extended-control mode, the method of checking protection had to be modified. The logical address of each block referenced is translated to locate each page and the associated key in storage. For future implementation of operands with a length greater than 256 bytes, such as MVCL, the protection checking routine will check as many keys in storage as required to successfully perform a protection check on an operand. This change is transparent to the user.

The only other change made to standard ASSIST-V to implement the Dynamic Address Translation feature was limiting the maximum length of an operand of psuedo instructions to 256 bytes. The size of the DAT concatenation buffers is fixed so an upper bound is placed on the length of operands. The System/370 instructions MVCL and CLCL are not supported in this version of ASSIST-V.

8. BUGS IN ASSIST-V AFFECTING DAT

There were two bugs found in ASSIST-V during the writing of the Dynamic Address Translation feature worthy of note. The first bug concerns protection checking of psuedo instructions. ASSIST-V always does protection checking based on information from the second byte of the

instruction. However, the length of a psuedo instruction is contained in the third byte. ASSIST-V checks the length of these instructions using the extended opcode in the second byte instead of the length. The failure to properly obtain the correct length of the operand can result in the failure to detect and concatenate an operand that spans across two pages.

A similar bug was found in the execution of the instructions TRANSLATE and TRANSLATE AND TEST. The length of the second operand in both of these instructions is fixed at 256 bytes. ASSIST-V does not recognize this and does a protection check based on whatever value is in the second byte of the instruction. This results in a protection check of the second operand based on the length of the first. As in the case of the psuedo instructions, this can result in the failure to detect and concatenate an operand which spans across two pages. Both of these bugs have been corrected in the current version of ASSIST-V.

9. POSSIBLE ADDITIONS TO DAT

The Dynamic Address Translation feature presently implemented assumes the existence of associative array registers in the translation process and in doing so assumes a negligible translation time. The System/370 maintains a set of special-purpose registers which contain the most recently referenced pages from the executing program, identified by their segment and page numbers. When a

reference is made to a logical address, each of the associative array registers are searched in parallel for a match on the referenced page. If a match is found, the address is translated completely within hardware with a very small time degradation due to address translation. At the same time the search of the associative array registers begins, translation begins through the translation tables.

A realistic illustration of the degradation of program execution time due to address translation could be demonstrated by simulating the associative array register process in ASSIST-V. A table of the last N referenced pages could be kept by the Dynamic Address Translation feature as a simulation of associative array registers. If the currently referenced page is present in this table, the user would not be charged for translation time; however, if the currently referenced page is not in the table, the user would be charged for two memory references in the case of a successful translation, two memory references and a page-in operation in the case of a page fault, and one memory reference and a page-in operation in case of a segment fault. Test programs could be run both in extended-control mode and basic-control mode to show the difference in execution time between virtual and non-virtual memory.

LIST OF REFERENCES

LIST OF REFERENCES

1. Donovan, John J., Systems Programming, McGraw-Hill Book Company, New York, 1972.
2. Gordon, Geoffrey, System Simulation, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1978.
3. Hughes, Charles E., and Pfleeger, Charles P., An Environment Simulator for IBM 360 Systems Software Development, Report CS-77-23, The University of Tennessee Computer Science Department, Knoxville, Tenn.
4. Hughes, Charles E., and Pfleeger, Charles P., ASSIST-V Program Logic Manual, Report CS-77-25, The University of Tennessee Computer Science Department, Knoxville, Tenn.
5. Hughes, Charles E., and Pfleeger, Charles P., ASSIST-V Users Guide, Report CS-77-24, The University of Tennessee Computer Science Department, Knoxville, Tenn.
6. IBM Introduction to Virtual Storage in System/370, IBM Corp., White Plains, N.Y.
7. IBM Operating System Job Control Language Reference, Form GC28-6704, IBM Corp., White Plains, N.Y.
8. IBM System/360 Operating System Assembler Language, Form GC28-6514, IBM Corp., White Plains, N.Y.
9. IBM System/360 Principles of Operation, Form GA22-6821, IBM Corp., White Plains, N.Y.
10. IBM System/370 Principles of Operation, Form GA22-7000, IBM Corp., White Plains, N.Y.
11. Madnick, Stuart E., and Donovan, John J., Operating Systems, McGraw-Hill Book Company, New York, 1974.
12. Mashey, John R., and Campbell, Graham, ASSIST User's Manual, Pennsylvania State University Computer Science Department, University Park, Penn.
13. Overbeek, Ross A., and Singletary, W. E., Assembler Language with ASSIST, Science Research Associates, Inc., Chicago, Ill., 1976.
14. Rindfleisch, Daniel H., Debugging System 360/370 Programs Using OS and VS Storage Dumps, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1976.

15. Rudd, Walter G., Assembly Language Programming and the IBM 360 and 370 Computers, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1976.
16. Tanenbaum, Andrew S., Structured Computer Organization, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1976.

VITA

Anthony Michael Peters was born in Wilton Junction, Iowa on June 26, 1952. He attended elementary schools in that city as well as Story City, Iowa. He attended high schools in St. Louis, Missouri and Nashville, Tennessee where he graduated from Glencliff High School in June 1970.

After a two-year tour of duty with the United States Marines, he entered The University of Tennessee, Nashville, in September 1972. He received a Bachelor of Arts degree in Mathematics in June 1978. In the spring of 1978 he began a graduate program in Computer Science at The University of Tennessee, Knoxville. In the fall of 1978 he accepted a teaching assistantship for the Computer Science Department at The University of Tennessee, Knoxville. He received a Master of Science degree in Computer Science in 1979.

He is married to the former Frances Rae Roach and has two children, Daphne Anne and Whitney Michael.